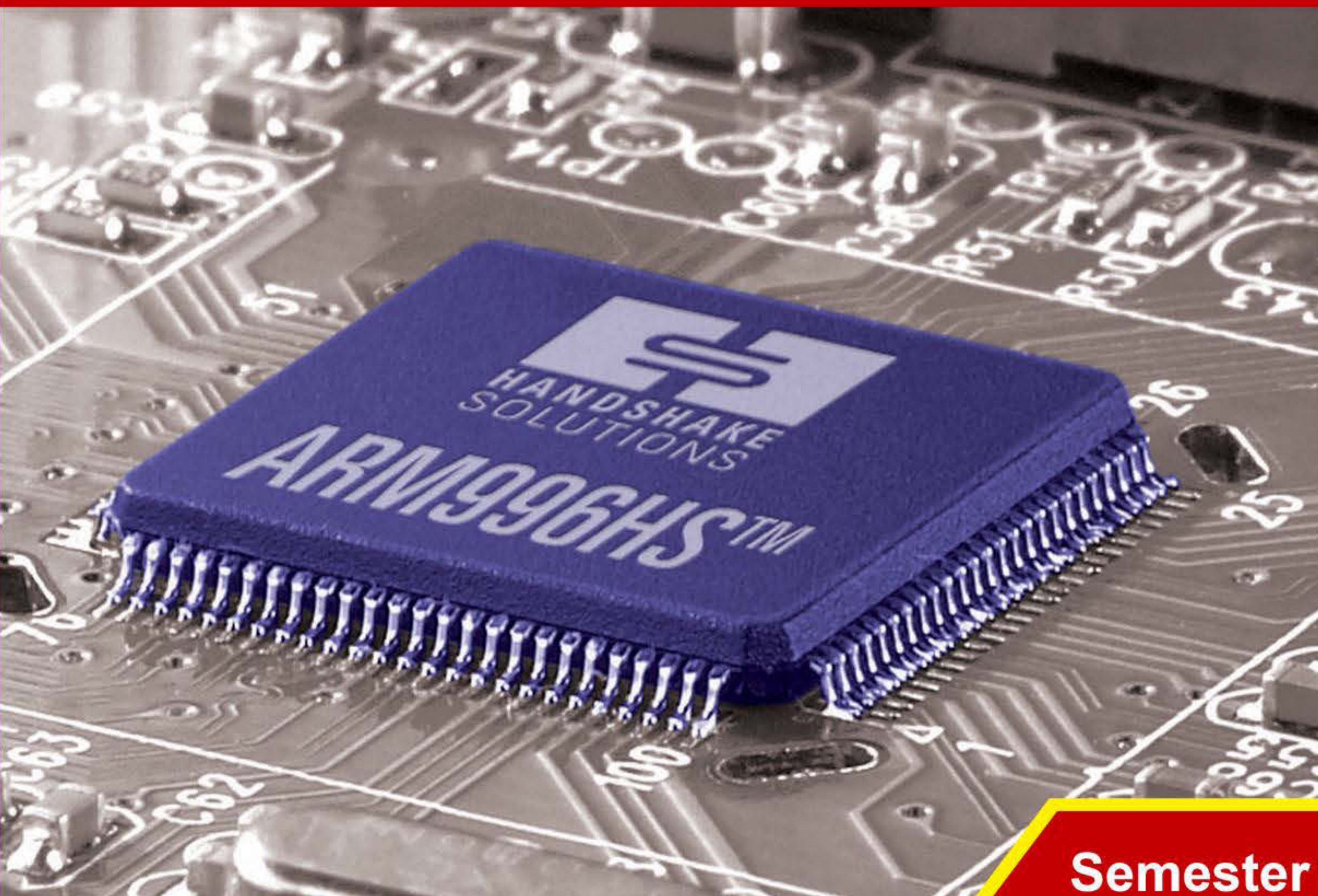
Kementerian Pendidikan Dan Kebudayaan
Republik Indonesia
2013

TEKNIK MIKROPROSESOR



Semester
2

Untuk SMK / MAK Kelas X

Penulis	**: DJOKO SUGIONO**
Editor Materi	**: WELDAN KHUSUF**
Editor Bahasa	**:**
Ilustrasi Sampul	**:**
Desain & Ilustrasi Buku	**: PPPPTK BOE MALANG**

Pusat Pengembangan & Pemberdayaan Pendidik & Tenaga Kependidikan Bidang Otomotif & Elektronika:

Jl. Teluk Mandar, Arjosari Tromol Pos 5, Malang 65102, Telp. (0341) 491239, (0341) 495849, Fax. (0341) 491342, Surel: vedcmalang@vedcmalang.or.id, Laman: www.vedcmalang.com

DISKLAIMER (DISCLAIMER)

Penerbit tidak menjamin kebenaran dan keakuratan isi/informasi yang tertulis di dalam buku tek ini. Kebenaran dan keakuratan isi/informasi merupakan tanggung jawab dan wewenang dari penulis.

Penerbit tidak bertanggung jawab dan tidak melayani terhadap semua komentar apapun yang ada didalam buku teks ini. Setiap komentar yang tercantum untuk tujuan perbaikan isi adalah tanggung jawab dari masing-masing penulis.

Setiap kutipan yang ada di dalam buku teks akan dicantumkan sumbernya dan penerbit tidak bertanggung jawab terhadap isi dari kutipan tersebut. Kebenaran keakuratan isi kutipan tetap menjadi tanggung jawab dan hak diberikan pada penulis dan pemilik asli. Penulis bertanggung jawab penuh terhadap setiap perawatan (perbaikan) dalam menyusun informasi dan bahan dalam buku teks ini.

Penerbit tidak bertanggung jawab atas kerugian, kerusakan atau ketidaknyamanan yang disebabkan sebagai akibat dari ketidakjelasan, ketidaktepatan atau kesalahan didalam menyusun makna kalimat didalam buku teks ini.

Kewenangan Penerbit hanya sebatas memindahkan atau menerbitkan mempublikasi, mencetak, memegang dan memproses data sesuai dengan undang-undang yang berkaitan dengan perlindungan data.

Katalog Dalam Terbitan (KDT)
Teknik Elektronika, Edisi Pertama 2013
Kementerian Pendidikan & Kebudayaan
Direktorat Jenderal Peningkatan Mutu Pendidik & Tenaga Kependidikan, th. 2013: Jakarta

KATA PENGANTAR

Puji syukur kami panjatkan kepada Tuhan yang Maha Esa atas tersusunnya buku teks ini, dengan harapan dapat digunakan sebagai buku teks untuk siswa Sekolah Menengah Kejuruan (SMK) Bidang Studi Keahlian Teknologi Dan Rekayasa, TEKNIK ELEKTRONIKA.

Penerapan kurikulum 2013 mengacu pada paradigma belajar kurikulum abad 21 menyebabkan terjadinya perubahan, yakni dari pengajaran (*teaching*) menjadi BELAJAR (*learning*), dari pembelajaran yang berpusat kepada guru (*teachers-centered*) menjadi pembelajaran yang berpusat kepada peserta didik (*student-centered*), dari pembelajaran pasif (*pasive learning*) ke cara belajar peserta didik aktif (*active learning-CBSA*) atau *Student Active Learning-SAL*.

Buku teks ″TEKNIK MIKROPROSESOR” ini disusun berdasarkan tuntutan paradigma pengajaran dan pembelajaran kurikulum 2013 diselaraskan berdasarkan pendekatan model pembelajaran yang sesuai dengan kebutuhan belajar kurikulum abad 21, yaitu pendekatan model pembelajaran berbasis peningkatan keterampilan proses sains.

Penyajian buku teks untuk Mata Pelajaran ″TEKNIK MIKROPROSESOR″ ini disusun dengan tujuan agar supaya peserta didik dapat melakukan proses pencarian pengetahuan berkenaan dengan materi pelajaran melalui berbagai aktivitas proses sains sebagaimana dilakukan oleh para ilmuwan dalam melakukan eksperimen ilmiah (penerapan scientifik), dengan demikian peserta didik diarahkan untuk menemukan sendiri berbagai fakta, membangun konsep, dan nilai-nilai baru secara mandiri.

Kementerian Pendidikan dan Kebudayaan, Direktorat Pembinaan Sekolah Menengah Kejuruan, dan Direktorat Jenderal Peningkatan Mutu Pendidik dan Tenaga Kependidikan menyampaikan terima kasih, sekaligus saran kritik demi kesempurnaan buku teks ini dan penghargaan kepada semua pihak yang telah berperan serta dalam membantu terselesaikannya buku teks siswa untuk Mata Pelajaran TEKNIK MIKROPROSESOR kelas X/Semester 2 Sekolah Menengah Kejuruan (SMK).

Jakarta, 12 Desember 2013
Menteri Pendidikan dan Kebudayaan

Prof. Dr. Mohammad Nuh, DEA

DAFTAR ISI



INSTRUKSI MIKROPROSESOR

DESKRIPSI MATERI PEMBELAJARAN

Sebuah *hardware* mikroprosesor tidak akan dapat berfungsi apabila tidak dilengkapi dengan instruksi-instruksi, karena melalui instruksi inilah komponen-komponen sebuah mikroprosesor dikendalikan. Agar mikroprosesor dapat digunakan untuk memenuhi keperluan dari kendali peralatan rumah tangga, sistem keamanan mobil dan rumah sampai pada sistem kendali di bidang militer dan bidang industri, diperlukan susunan instruksi secara sekuensial sehingga dapat berfungsi sebagai penggerak fungsi kendali. Sedangkan instruksi sistem mikroprosesor dituliskan dalam bahasa yang disebut dengan bahasa assembly, sehingga dengan memahami instruksi dan menyusunnya dalam urutan sekuensial akan dapat digunakan Untuk dapat menerapkan instruksi pada sistem mikroprosesor perlu memahami deskripsi setiap instruksi, operasi untuk menunjang fungsi instruksi terkait dengan sistem hardware mikroprosesor.

KOMPETENSI INTI (KI-3)

Kompetensi Dasar (KD):

1. Menyajikan instruksi dalam bahasa assembly mikroprosesor

Indikator:

1.1. Memahami instruksi dalam bahasa *assembly*
1.2. Memahami urutan penggunaan instruksi bahasa assembly

KOMPETENSI INTI (KI-4)

Kompetensi Dasar (KD):

1. Menerapkan instruksi dalam bahasa assembly

Indikator:

1.1. Melakukan eksperimen untuk membuktikan penggunaan masing-masing instruksi bahasa assembly.
1.2. Melakukan eksperimen dengan menggunakan instruksi bahasa assembly dan mengaplikasikannya kedalam suatu kasus keteknikan

KATA KUNCI PENTING

- Instruksi
- bahasa assembly

BAB I. INSTRUKSI MIKROPROSESOR

1.1. DASAR KOMPONEN MIKROPROSESOR

Seperti dijelaskan pada bab sebelumnya, bahwa alam semesta, semua mahluk baik binatang ataupun tumbuhan dan manusia merupakan ciptaan Tuhan Yang Maha Kuasa, dari sekian banyak ciptaanNYA coba kita simak tubuh manusia.

Ditinjau dari struktur dan fungsinya tubuh manusia terdiri dari berbagai macam organ misal jantung, hati, kepala, kaki, tangan dan organ tubuh lainnya yang kesemuanya membentuk sebuah sistem tubuh, setiap komponen atau organ tubuh tersebut memiliki fungsi masing-masing dalam membentuk sistem tubuh dan dapat bekerja saling berhubungan satu ama lainnya.

Bagaimana proses dan dari mana sumber yang membuat setiap gerak atau kerja organ, dunia medis menjelaskan bahwa kerja atau gerak organ tubuh manusia selalu diawali adanya instruksi atau perintah dari otak berupa sinyal listrik yang dikirimkan melalui jalur kendali menuju organ yang diperintahkan untuk melaksanakan gerak. Setiap kerja atau gerak organ melalui sinyal peritah dan gerak organ satu dengan lainnya juga selalu ada sinkronisasi.

Tanpa adanya instruksi atau perintah dari otak maka organ yang dibentuk dari sel tidak bekerja, begitu juga jika ada kegagalan instruksi dari otak akan membuat organ tidak bekerja sesuai yang diharapkan misal terjadi pada orang sakit stroke. Seperti diketahui organ tubuh manusia mampu bekerja selama 24 jam terus menerus, dari kenyataan tentang kerja organ tubuh manusia maka sudah seharusnya kita bersyukur dan mau mengakui keagunganNYA.

Dalam bab sebelumnya juga dijelaskan tentang arsitektur sistem mikroprosesor yang dikembangkan mengikuti pola tubuh manusia, yaitu sistem dibuat dengan membentuk organ-organ yang sering istilahnya disebut dengan komponen. Setiap komponen pembentuk sistem mikroprosesor dibangun dari rangkaian-rangkaian gerbang digital, antar komponen dihubungkan dengan sistem jalur data (*Data BUS*), untuk menunjuk komponen digunakan jalur alamat (*Address Bus*) dan untuk sinkronisasi kerja antar komponen digunakan jalur kontrol (*Control Bus*). Seperti pada tubuh manusia pada sistem mikroprosesor untuk

memanfaatkan atau memfungsi komponen dilakukan melalui instruksi atau perintah yang dikirim dari pusat pengolah yang disebut *Arithmatic Logic Unit.*

Untuk memahami instruksi-instruksi sistem mikroprosesor, berikut dijelaskan pengertian instruksi, deskripsi instruksi dan operasional mikroprosesor Z80.

Dalam kamus besar bahasa indonesia Instruksi diartikan sebagai perintah atau arahan untuk melakukan suatu pekerjaan atau melaksanakan suatu tugas, memberi instruksi berarti memberi perintah atau arahan yang di dalamnya terdapat aturan-aturan teknis, prosedur dan capaian pelaksanaan instruksi yang diharapkan.

Pengertian instruksi atau perintah dalam bahasa asembler terdiri dari susunan kode biner yang membentuk tata cara dan tata kelola operasi hardware mikroprosesor, kode biner digunakan sebagai dasar pembentuk instruksi dengan alasan bahwa sebuah mikroprosesor secara hardware hanya mengenali dua kondisi yaitu nol dan satu. Seperti dijelaskan pada bab sebelumnya bahwa secara struktur mikroprosesor dibangun dari sekian banyak transistor yang berfungsi sebagai saklar elektronik, sedangkan saklar elektronik hanya kenal on /off atau buka/tutup.

Banyaknya instruksi per satuan waktu biasanya diukur dalam detik (*instructions per second (IPS)*) adalah sebuah ukuran kecepatan proses dari prosesor sebuah komputer. IPS ditentukan berdasarkan kecepatan maksimum yang dicapai oleh sebuah prosesor dalam melaksanakan instruksi untuk setiap detiknya, hal tersebut juga dijadikan patokan pengukuran kecepatan operasi sebuah aplikasi karena antara satu aplikasi dengan aplikasi lainnya memiliki kecepatan yang berbeda.

Bagi pengguna bahasa mesin dengan kode biner sangat merepotkan, untuk lebih sederhana dalam mengenali instruksi dengan bilangan biner dikodekan ke dalam bilangan heksadesimal yang dikenal dengan bahasa mesin.

Dengan demikian dapat mempermudah bagi pengguna untuk mengenali instruksi yang digunakan, namun demikian masih banyak kendala dalam melakukan intepretasi terhadap kode instruksi tersebut yang disebabkan beberapa instruksi ternyata dikodekan lebih dari satu bilangan heksadesimal.

Pada akhirnya disimbolkan dalam bentuk mnemonic untuk bisa dipahami fungsi dan operasinya oleh programer. Berikut merupakan penjelasan fungsi dan kode instruksi serta cara penulisannya:

SLA reg_8

<table>
<tr><td>Operasi</td><td>Isi dari reg_8 posisinya digeser ke kiri 1 bit, Isi bit 7 disalin ke carry flag dan isi dari zero dimasukan ke bit 0.</td></tr>
<tr><td>Op Code</td><td>11001011 : 00100[reg_8]
<table>
<tr><th>Register</th><th>Bit</th></tr>
<tr><td>A</td><td>111</td></tr>
<tr><td>B</td><td>000</td></tr>
<tr><td>C</td><td>001</td></tr>
<tr><td>D</td><td>010</td></tr>
<tr><td>E</td><td>011</td></tr>
<tr><td>H</td><td>100</td></tr>
<tr><td>L</td><td>101</td></tr>
<tr><td>(HL)</td><td>110</td></tr>
</table></td></tr>
<tr><td>Flag</td><td>S Z berubah sesuai yang diinginkan (ditentukan)
H N di-reset
P/V merupakan paritas</td></tr>
</table>

Mikroprosesor Z80 CPU dapat mengoperasionalkan 158 macam tipe instruksi termasuk 78 buah instruksi yang dimiliki mikroprosesor 8080A CPU. Di dalam mikroprosesor Z80 beberapa macam instruksi tersebut yang dikelompokan berdasarkan fungsinya, yang meliputi :

- Mengisi dan Menukar



- Transfer Blok
- Arithmatika dan logika
- Putar dan Geser
- Bit Manipulasi (Set, Reset, Test)
- Jump, Call, dan Return
- Input/Output
- Dasar Kontrol CPU

Contoh penulisan beberapa instruksi:

Instruksi	Siklus	Op-code	Ukuran
ADC A,(HL)	7	8E	1
ADC A,(IX+o)	19	DD 8E oo	3
ADC A,(IY+o)	19	FD 8E oo	3
ADC A,n	7	CE nn	2
ADC A,r	4	88+r	1
ADC A,IXp	8	DD 88+p	2
ADC A,IYq	8	FD 88+q	2
ADC HL,BC	15	ED 4A	2
ADC HL,DE	15	ED 5A	2
ADC HL,HL	15	ED 6A	2
ADC HL,SP	15	ED 7A	2
ADD A,(HL)	7	86	1
ADD A,(IX+o)	19	DD 86 oo	3
ADD A,(IY+o)	19	FD 86 oo	3
ADD A,n	7	C6 nn	2
ADD A,r	4	80+r	1
ADD A,IXp	8	DD 80+p	2
ADD A,IYq	8	FD 80+q	2
ADD HL,BC	11	09	1
ADD HL,DE	11	19	1
ADD HL,HL	11	29	1

1. Instruksi Mengisi dan Menukar

Instruksi mengisi dan menukar isi dari register dan atau memori dibutuhkan pengalamatan yang diikut sertakan ke dalam setiap instruksi, alamt tersebut digunakan untuk menunjuk sumber data dan tujuan data yang diinginkan. Sebagai contoh, mengisi register C dari register B menggunakan op-code 48H, dalam bilangan biner adalah 0100 1000. sedangkan mnemonic bahasa assembly

untuk seluruh kelompok ini adalah LD, diikuti dengan tujuan, selanjut-nya diikuti sumber data seperti format berikut:

LD tujuan , sumber.

Beberapa kombinasi mode pengalamatan yang mungkin dilakukan, misalnya sumber menggunakan pengalamatan serti dalam daftar alamat dan tujuan dengan register tidak langsung, misal mengisi lokasi memori yang ditunjuk oleh register HL dengan isi register D. Op-code untuk operasi ini adalah 72H, dan mnemonic untuk instruksi mengisi adalah:

LD (HL) , D

Tanda kurung di sekitar HL menunjukkan bahwa isi dari HL digunakan sebagai pointer ke lokasi memori . Dalam mnemonik semua instruksi Z80 untuk mengisi dan menukar tujuan selalu ditrulis pertama dan diikuti dengan sumber. Dalam bahasa assembly mikroprosesor Z80 untuk kemudahan pemrograman setiap instruksi didefinisikan sebagai bagian kecil dalam program, hal ini dilakukan untuk mempermudah bagi programer untuk mengenali setiap instruksi dalam mikroprosesor Z80.

Instruksi Data, Instruksi data merupakan instruksi yang terkait dengan pemindahan data secara langsung antar register atau register dengan memori, misal instruksi data MOV, dengan mnemonik MOV BC, DE mempunyai arti mengisi register BC dengan data yang berada di DE.

Beberapa Op-code yang tersedia di Z80 menggunakan dua byte, dengan fitur ini merupakan metode yang efisien dalam pemanfaatan memori.

2. Instruksi Transfer blok

Instruksi transfer blok yang sangat baik untguk memindahkan sejumlah data, instruksi beroperasi dengan tiga register, yaitu:

1. HL penunjuk lokasi sumber
2. DE penunjuk lokasi tujuan
3. BC adalah byte counter

Setelah programer menginisialisasi tiga register tersebut, salah satu dari keempat instruksi LDI (Load dan Increment) dapat digunakan, dan sebagai penunjuk alamat yang bergerak satu byte penunjuk lokasi asal data oleh HL dan penunjuk lokasi tujuan oleh DE.

Register pasangan HL dan DE kemudian secara otomatis bertambah dan siap untuk menunjuk ke lokasi-lokasi berikutnya, byte counter yang diwakili register BC juga dikurangi 1(satu) atau dekrimen.

Instruksi ini sangat berharga ketika satu blok data harus dipindahkan dari satu blok lokasi ke satu blok lokasi lainnya, proses terus berjalan dan data bergerak dari lokasi yang ditunjuk HL menuju lokasi yang ditunjuk DE dengan jumlah data yang dipindahkan sejumlah BC.

Instruksi pemindahan blok data pada hakekatnya merupakan operasi inkremen diulang sampai byte counter mencapai menghitung dari nol, dengan demikian instruksi tunggal ini dapat memindahkan satu blok data dari satu lokasi ke lokasi yang lain . Oleh karena yang digunakan merupakan register 16-bit, maka ukuran blok data yang bisa sampai dipindahkan mencapai 64 Kbytes (1K = 1024) panjang dan dapat dipindahkan dari lokasi manapun di memori ke lokasi lain.

3. Instruksi Arithmatika dan Logika

ALU, singkatan dari *Arithmetic And Logic Unit* (unit aritmatika dan logika), adalah bagian mikroprosesor yang berfungsi melakukan operasi hitungan aritmatika dan logika, operasi aritmatika meliputi operasi penjumlahan dan pengurangan.Sedangkan operasi logika yang dapat diloakukan oleh ALU meliputi logika AND dan OR.

ALU melakukan operasi aritmatika pengurangan, perkalian dan pembagian dilakukan dengan dasar penjumlahan (dalam bahasan bab sebelumnya), instruksi ALU lainnya adalah melakukan keputusan dari operasi logika sesuai instruksi yang diberikan, Instruksi operasi logika meliputi operasi logika dari dua buah elemen logika menggunakan operator logika

Sebagai contoh untuk instruklsi DEC atau instruksi INC yang merupakan instruksi beroperasi dengan 8-bit, dan operasinya dilakukan antara data yang berada di akumulator dan sumber data serta hasil operasi ditempatkan di akumulator.

Pengecualian untuk operasi membandingkan (CP) antara dua buah data, maka akumulator tidak berubah dan semua operasi ini mempengaruhi register flag sebagai hasil dari operasi. instruksi INC dan DEC.

Akumulator sebelum operasi	1111	0011	=	F3H
Operan	0000	0111	=	07H
Hasil sampai Akumulator	0000 0011 = 03H			

Fungsi-fungsi yang didefinisikan pada ALU adalah *Add* (penjumlahan), *Addu* (penjumlahan tidak bertanda), *Sub* (pengurangan), *Subu* (pengurangan tidak bertanda), and, or, xor, sll (*shift left logical*), srl (*shift right logical*), sra (*shift right arithmetic*), dan lain-lain.

Operasi dasar aritmetika adalah penjumlahan, pengurangan, perkalian dan pembagian, demikian juga untuk operasi-operasi lain yang menuntut persyaratan lebih kompleks sepertii akar, pangkat, persentase, dan algoritma.

Proses perhitungan dalam aritmetika dilakukan secara sekuensial berdasarkan urutan operasinya, artinya proses ditentukan sesuai dengan aturan operasi aritmetika mana dilakukan lebih dulu dan mana yang berikutnya. Termasuk di dalamnya adalah aritmetika bilangan asli, bilangan bulat, bilangan rasional, dan bilangan real.

Instruksi Kontrol, Instruksi kontrol adalah instruksi yang memutuskan alur dari sekuen instruksi yang akan dieksekusi. Contoh instruksi kontrol adalah JMP

Program komputer atau sering kali disingkat sebagai program adalah serangkaian instruksi yang ditulis untuk melakukan suatu fungsi spesifik ...

4. Instruksi Putar dan geser

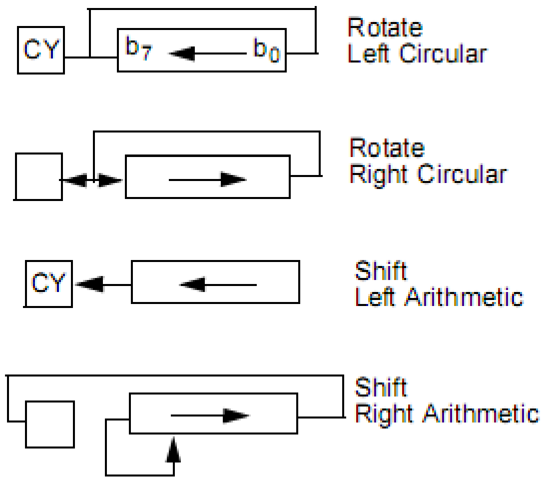
Fitur utama dari instruksi putar dan geser pada mikroprosesor Z80 adalah untuk memutar atau menggeser data dalam akumulator, atau data yang berada dalam setiap register general-purpose, atau lokasi memori.

Instruksi putar dapat diterapkan untuk memutar data dalam register atau memori ke arah kiri atau ke arah kanan, dan untuk instruksi putar dapat melibatkan carry pada flag atau hanya terbatas dalam register itu sendiri.

Sedangkan instruksi geser pada prinsipnya juga hampir sama, yaitu menggeser data dalam register atau memori ke arah kiri atau ke arah kanan, dan untuk instruksi geser dapat melibatkan carry pada flag atau hanya terbatas dalam register itu sendiri.

Instruksi geser data digeser bit per bit, sedangkan pada instruksi putar data dalam register digeser sebanyak 8 bit untuk satu kali instruksi putar.

Berikut merupakan diagram aliran data pada operasi geser atau putar:



Gambar 4.1. Diagram operasi instruksi geser dan instruksi putar

5. Instruksi Manipulasi Bit

Kemampuan untuk set, reset, dan menguji bit individu dalam register atau memori lokasi yang dibutuhkan di hampir setiap program. Manipulasi bit memungkinkan untuk mengubah isi bit dalam register flag, yaitu dengan tujuan fasilitasi perangkat lunak untuk pemanggilan rutin tertentu, indikasi kontrol kondisi eksternal, atau data dikemas ke lokasi memori, membuat pemanfaatan memori yang lebih efisien.

Instruksi manipulasi bit dalam mikroprosesor Z80 dapat men-set, reset, atau menguji bit dalam akumulator, setiap register umum atau lokasi memori melalui sebuah instruksi tunggal.

Terdapat sejumlah 240 buah instruksi yang mendukung untuk tujuan ini.

Register yang dipilih meliputi akumulator atau register umum, yaitu register yang akan digunakan sebagai target operasi instruksi. Melalui pengalamatan register tidak langsung dan pengalamatan indeks memungkinkan untuk operasi instruksi dengan target lokasi memori eksternal.

Operasi uji bit (bit test) akan memberikan hasil set bit Zero pada register flag (Z), yaitu jika hasil uji bit ternyata sama dengan nol (zero).

6. Instruksi Jump, Call, dan Return

Instruksi ini berfungsi untuk melayani permintaan program untuk melakukan lompat dari satu alamat ke alamat tertentu, panggilan untuk sebuah rutin, dan kembali ke program utama saat rutin yang dipanggil telah selesai tugasnya.

Instruksi melompat yang dilaksanakan di Z80 CPU adalah sebuah cabang di dalam program, dimana program counter 16-bit memenuhi syarat seperti yang ditentukan oleh salah satu dari tiga yang tersedia mode pengalamatan (segera diperpanjang, relatif, atau langsung).

Perhatikan bahwa kelompok instruksi lompat memiliki beberapa kondisi yang dapat ditentukan sebelum lompat, jika kondisi ini tidak terpenuhi, program hanya berlanjut dengan instruksi sekuensial berikutnya. Kondisi untuk lompat tergantung pada bit data dalam register flag.

Pengalamatan langsung diperpanjang digunakan untuk melompat ke setiap lokasi di memori,. instruksi ini membutuhkan tiga byte (dua untuk menentukan alamat 16-bit) dengan urutan rendah alamat byte pertama, diikuti dengan alamat urutan tinggi byte.

Sebagai contoh, sebuah lompatan tanpa syarat ke lokasi memori 3E32H adalah:

Address A	C3	Op Code
A+1	32	Low Order Address
A+2	3E	High Order Address

Instruksi jump relatif hanya menggunakan dua byte, perpindahan bisa di kisaran 129 sampai -126 dan diukur dari alamat dimana Op Code instruksi berada.

Terdapat tiga jenis register untuk fasilitas instruksi lompat tidak langsung, yaitu pasangan register HL atau salah satu dari Indeks register 1X atau IY yang isinya langsung dipindahkan ke register PC. Fitur ini memungkinkan untuk digunakan dalam program lompat sebagai fungsi.

7. Instruksi Input / Output

Mikroprosesor Z80 memiliki serangkaian instruksi input dan output, pengalamatan dari perangkat input atau output dapat berupa absolut atau langsung, menggunakan register C. Dalam register mode pengalamatan tidak langsung, data dapat ditransfer antara perangkat I /O dan salah satu register internal. Selain itu delapan blok instruksi transfer telah dilaksanakan, instruksi-instruksi ini mirip dengan transfer blok memori.

Pengalamatan menggunakan pasangan HL untuk pointer ke sumber memori (perintah output) atau tujuan (input perintah) sedangkan register B digunakan sebagai byte counter. Register C memegang alamat port input atau output yang digunakan, register B adalah penyimpan data delapan bit, dan untuk perintah transfer blok I/O menangani hingga 256 byte.

IN A, merupakan instruksi baca data dari port I/O dengan alamat A dan OUT n, A, keluarkan data dari akumulator ke port I/O beralamat n.

8. Instruksi Kelompok Kontrol CPU.

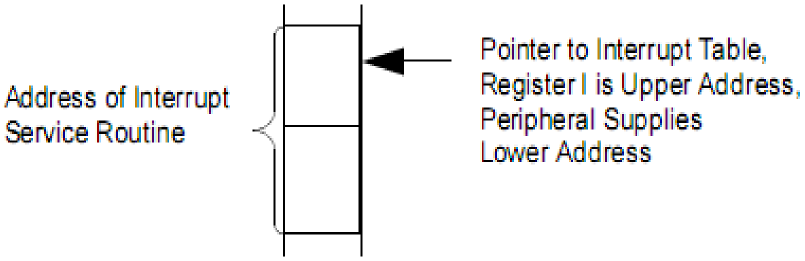
Beberapa tujuan secara umum instruksi kontrol CPU , meliputi instruksi NOP merupakan instruksi yang tidak melakukan apa-apa, Instruksi HALT menghentikan CPU sementara, DI dan EI adalah operasi interupsi yang diterima digunakan untuk mengunci atau mengaktifkan interupsi .

Terkait dengan interupsi terdapat tiga modus interrupsi, yaitu perintah set CPU untuk memilih ke salah satu dari tiga respon interrupsi tersedia dan ketiga mode dapat dijelaskan sebagai berikut:

Mode 0, perangkat yang menginterupsi dapat menyisipkan instruksi pada bus data dan memungkinkan CPU untuk mengeksekusinya .

Mode 1, modus yang disederhanakan di mana CPU secara otomatis mengeksekusi restart (RST) ke 0038H, sehingga tidak ada hardware eksternal diperlukan (isi PC tetap dan diarahkan ke stack) .

Mode 2, modus yang paling kuat karena memungkinkan untuk panggilan tidak langsung untuk setiap lokasi di memori . Dengan mode in CPU membentuk alamat memori 16 - bit di mana bagian delapan bit atas adalah isi register I dan delapan bit rendah dipasok oleh perangkat yang menginterupsi. Seperti ditunjukan pada gambar berikut



Gambar 4.2. Layanan interupsi dalam pelaksanaan program
(instruksi manual Z80, Zilog)

1.2. DESKRIPSI INSTRUKSI MIKROPROSESOR Z80.

Seperti dijelaskan pada bab sebelumnya bahwa mikroprosesor Z80 memiliki sekumpulan instruksi yang memiliki fungsi operasional dan aktivasi hardware, terdapat beberapa instruksi seperti pemindahan blok untuk transfer data yang

cepat dan efisien dalam memori atau antara memori dan I/O.

Demikian juga instruksi mikroprosesor Zilog Z80 dibagi ke dalam kategori pemuatan 8-bit, pemuatan 16-bit, pertukaran, transfer blok, pencarian, operasi logika dan aritmatika 8-bit, kontrol CPU Perputaran (rotasi) dan pergeseran (shift), operasi set, reset dan tes bit, lompatan, pemanggilan (*call*), kembali (*return*) dan restart, serta Operasi masukan dan keluaran.

Berikut merupakan penjelasan setiap instruksi yang dapat dioperasionalkan pada mikroprosesor Z80, meliputi deskripsi fungsi instruksi, op-code, dampak opeasi terhadap flag dan state yang diperlukan untuk pelaksanaan sebuah instruksi tersebut.

1. Instruksi Transfer Data Z80.

Instruksi transfer data merupakan instruksi yang digunakan oleh mikroprosesor Z80 untuk memindahkan data dari satu register ke register lainnya, dari satu register ke suatu lokasi memori atau memindahkan data dari satu lokasi memori ke lokasi memori yang lain atau dari memori ke register.

Untuk instruksi transfer ini meliputi instruksi **EX, EXX, LD, LDD, LDDR, LDI, LDIR, POP, PUSH.**

a. Instruksi EX

EX DE,HL

Operasi	pertukaran 16-bit antara isi register DE dan HL.
Op Code	11101011
T State	4

EX AF,AF'

Operasi	pertukaran 16-bit antara isi register AF dan AF'.

Op Code	00001000
T State	4

EX (SP),HL

Operasi	pertukaran (SP) dengan L, dan (SP+1) dengan H.
Op Code	11100011
T State	19

EX (SP),reg$_{index}$

<table>
<tr><td>Operasi</td><td>pertukaran (SP) dengan LSB dari reg$_{index}$, dan (SP+1) dengan MSB dari reg$_{index}$.</td></tr>
<tr><td>Op Code</td><td>[reg$_{index}$] : 11100011
<table>
<tr><td>Register</td><td>Bit</td></tr>
<tr><td>IX</td><td>11011101</td></tr>
<tr><td>IY</td><td>11111101</td></tr>
</table></td></tr>
<tr><td>T State</td><td>23</td></tr>
</table>

b. Instruksi EXX

EXX

Operasi	pertukaran 16-bit isi dari BC, DE, dan HL dengan BC', DE', dan HL'.
Op Code	11011001
T State	4

c. Instruksi LD

LD reg$_8$D,reg$_8$S

<table>
<tr><td>Operasi</td><td>Isi dari register reg$_8$S disimpan ke dalam reg$_8$D.</td></tr>
<tr><td>Op Code</td><td>01[reg$_8$D][reg$_8$S]
<table>
<tr><th>Register</th><th>Bit</th></tr>
<tr><td>A</td><td>111</td></tr>
<tr><td>B</td><td>000</td></tr>
<tr><td>C</td><td>001</td></tr>
<tr><td>D</td><td>010</td></tr>
<tr><td>E</td><td>011</td></tr>
<tr><td>H</td><td>100</td></tr>
<tr><td>L</td><td>101</td></tr>
<tr><td>(HL)</td><td>110</td></tr>
</table></td></tr>
<tr><td>T State</td><td>4 atau 7 (HL)</td></tr>
</table>

LD reg$_8$,imm$_8$

<table>
<tr><td>Operasi</td><td>Menyimpan nilai immediate ke register reg$_8$.</td></tr>
<tr><td>Op Code</td><td>00[reg$_8$]110 : [imm$_8$]
<table>
<tr><th>Register</th><th>Bit</th></tr>
<tr><td>A</td><td>111</td></tr>
<tr><td>B</td><td>000</td></tr>
<tr><td>C</td><td>001</td></tr>
<tr><td>D</td><td>010</td></tr>
<tr><td>E</td><td>011</td></tr>
<tr><td>H</td><td>100</td></tr>
</table></td></tr>
</table>

	<table><tr><td>L</td><td>101</td></tr><tr><td>(HL)</td><td>110</td></tr></table>
T State	7 atau 10 (HL)

LD reg_8,(reg_{index} + ofs_8)

Operasi	Menyimpan nilai ditunjuk oleh reg_{index} ditambah ofs_8 ke dalam reg_8.
Op Code	[reg_{index}] : 01[reg_8]110 : [ofs_8] <table><tr><td>**Register**</td><td>**Bit**</td></tr><tr><td>IX</td><td>11011101</td></tr><tr><td>IY</td><td>11111101</td></tr><tr><td>**Register**</td><td>**Bit**</td><td></td></tr><tr><td>A</td><td>111</td><td></td></tr><tr><td>B</td><td>000</td><td></td></tr><tr><td>C</td><td>001</td><td></td></tr><tr><td>D</td><td>010</td><td></td></tr><tr><td>E</td><td>011</td><td></td></tr><tr><td>H</td><td>100</td><td></td></tr><tr><td>L</td><td>101</td><td></td></tr></table>
T State	19

LD (reg_{index} + ofs_8),reg_8

<table>
<tr><td>Operasi</td><td>Menyimpan reg_8 ke lokasi memori yang ditunjuk oleh reg_{index} ditambah ofs_8.</td></tr>
<tr><td>Op Code</td><td>reg_{index} : 01110[reg_8] : [ofs_8]
<table>
<tr><th>Register</th><th>Bit</th></tr>
<tr><td>IX</td><td>11011101</td></tr>
<tr><td>IY</td><td>11111101</td></tr>
</table>
<table>
<tr><th>Register</th><th>Bit</th></tr>
<tr><td>A</td><td>111</td></tr>
<tr><td>B</td><td>000</td></tr>
<tr><td>C</td><td>001</td></tr>
<tr><td>D</td><td>010</td></tr>
<tr><td>E</td><td>011</td></tr>
<tr><td>H</td><td>100</td></tr>
<tr><td>L</td><td>101</td></tr>
</table>
</td></tr>
<tr><td>T State</td><td>19</td></tr>
</table>

LD (reg_{index} + ofs_8), imm_8

<table>
<tr><td>Operasi</td><td>Menyimpan data immediate ke lokasi memori yang ditunjuk oleh reg_{index} ditambah ofs_8.</td></tr>
<tr><td>Op Code</td><td>[reg_{index}] : 00110110 : [ofs_8] : [imm_8]
<table>
<tr><th>Register</th><th>Bit</th></tr>
<tr><td>IX</td><td>11011101</td></tr>
<tr><td>IY</td><td>11111101</td></tr>
</table>
</td></tr>
<tr><td>T State</td><td>19</td></tr>
</table>

LD A,(reg$_{16}$)

<table>
<tr><td>Operasi</td><td>Menyimpan isi memori yang ditunjuk oleh reg$_{16}$ ke dalam A.</td></tr>
<tr><td>Op Code</td><td>000[reg$_{16}$]1010
<table>
<tr><th>Register</th><th>Bit</th></tr>
<tr><td>BC</td><td>0</td></tr>
<tr><td>DE</td><td>1</td></tr>
</table></td></tr>
<tr><td>T State</td><td>7</td></tr>
</table>

LD A,(imm$_{16}$)

Operasi	Menyimpan isi memori yang ditunjuk oleh imm$_{16}$ ke dalam A.
Op Code	00111010 : [imm$_{LSB}$] : [imm$_{MSB}$]
T State	13

LD (reg$_{16}$),A

<table>
<tr><td>Operasi</td><td>Menyimpan A ke dalam memori yang ditunjuk oleh reg$_{16}$.
<table>
<tr><th>Register</th><th>Bit</th></tr>
<tr><td>BC</td><td>0</td></tr>
<tr><td>DE</td><td>1</td></tr>
</table></td></tr>
<tr><td>Op Code</td><td>000[reg$_{16}$]0010</td></tr>
<tr><td>T State</td><td>7</td></tr>
</table>

LD (imm$_{16}$),A

Operasi	Menyimpan A ke dalam memori yang ditunjuk oleh imm$_{16}$.
Op Code	00110010 : [imm$_{LSB}$] : [imm$_{MSB}$]
T State	13

LD A,{ I | R }

<table>
<tr><td>Operasi</td><td>Menyimpan isi register I atau R ke dalam A</td></tr>
<tr><td>Op Code</td><td>11101101 : 0101[reg]111
<table>
<tr><th>Register</th><th>Bit</th></tr>
<tr><td>I</td><td>0</td></tr>
<tr><td>R</td><td>1</td></tr>
</table></td></tr>
<tr><td>Flag</td><td>S Z berubah sesuai yang diinginkan (ditentukan).
H N di-reset.
P/V pegang IFF2.
C tidak berubah.</td></tr>
<tr><td>T State</td><td>9</td></tr>
</table>

LD { I | R },A

<table>
<tr><td>Operasi</td><td>Menyimpan nilai A ke dalam register I atau R.</td></tr>
<tr><td>Op Code</td><td>11101101 : 0100[reg]111
<table>
<tr><th>Register</th><th>Bit</th></tr>
<tr><td>I</td><td>0</td></tr>
<tr><td>R</td><td>1</td></tr>
</table></td></tr>
<tr><td>T State</td><td>9</td></tr>
</table>

LD reg_{16},imm_{16}

<table>
<tr><td>Operasi</td><td>Menyimpan data immediate ke dalam register reg_{16}.</td></tr>
<tr><td>Op Code</td><td>00[reg_{16}]0001 : [imm_{LSB}] : [imm_{MSB}]
<table>
<tr><th>Register</th><th>Bit</th></tr>
<tr><td>BC</td><td>00</td></tr>
<tr><td>DE</td><td>01</td></tr>
</table></td></tr>
</table>

<table>
<tr><td></td><td><table><tr><td>HL</td><td>10</td></tr><tr><td>SP</td><td>11</td></tr></table></td></tr>
<tr><td>T State</td><td>10</td></tr>
</table>

LD reg_{index},imm_{16}

<table>
<tr><td>Operasi</td><td>Menyimpan data immediate ke dalam register reg_{index}</td></tr>
<tr><td>Op Code</td><td>[reg_{index}] : 00100001 : [imm_{LSB}] : [imm_{MSB}]<table><tr><td>Register</td><td>Bit</td></tr><tr><td>IX</td><td>11011101</td></tr><tr><td>IY</td><td>11111101</td></tr></table></td></tr>
<tr><td>T State</td><td>14</td></tr>
</table>

LD HL,(imm_{16})

Operasi	Menyimpan isi memori yang ditunjuk oleh imm_{16} ke dalam HL.
Op Code	00101010 : [imm_{LSB}] : [imm_{MSB}]
T State	16

LD reg_{16},(imm_{16})

<table>
<tr><td>Operasi</td><td>Menyimpan isi memori yang ditunjuk oleh imm_{16} ke dalam register reg_{16}.</td></tr>
<tr><td>Op Code</td><td>11101101 : 01[reg_{16}]1011 : [imm_{LSB}] : [imm_{MSB}]<table><tr><td>Register</td><td>Bit</td></tr><tr><td>BC</td><td>00</td></tr><tr><td>DE</td><td>01</td></tr><tr><td>HL (see Undocumented)</td><td>10</td></tr><tr><td>SP</td><td>11</td></tr></table></td></tr>
</table>

T State	20

LD reg_{index},(imm_{16})

<table>
<tr><td>Operasi</td><td>Menyimpan isi memori yang ditunjuk oleh imm_{16} ke dalam index register reg_{index}.</td></tr>
<tr><td>Op Code</td><td>[reg_{index}] : 00101010 : [imm_{LSB}] : [imm_{MSB}]
<table>
<tr><th>Register</th><th>Bit</th></tr>
<tr><td>IX</td><td>11011101</td></tr>
<tr><td>IY</td><td>11111101</td></tr>
</table></td></tr>
<tr><td>T State</td><td>20</td></tr>
</table>

LD (imm_{16}),HL

Operasi	Menyimpan isi HL ke dalam memori yang ditunjuk oleh imm_{16}.
Op Code	00100010 : [imm_{LSB}] : [imm_{MSB}]
T State	16

LD (imm_{16}),reg_{16}

<table>
<tr><td>Operasi</td><td>Menyimpan isi register reg_{16} ke dalam memori yang ditunjuk oleh imm_{16}.</td></tr>
<tr><td>Op Code</td><td>11101101 : 01[reg_{16}]0011 : [imm_{LSB}] : [imm_{MSB}]
<table>
<tr><th>Register</th><th>Bit</th></tr>
<tr><td>BC</td><td>00</td></tr>
<tr><td>DE</td><td>01</td></tr>
<tr><td>HL</td><td>10</td></tr>
<tr><td>SP</td><td>11</td></tr>
</table></td></tr>
</table>

T State	20

LD (imm$_{16}$),reg$_{index}$

Operasi	Menyimpan isi index register reg$_{index}$ ke dalam meori yang ditunjuk oleh imm$_{16}$.
Op Code	[reg$_{index}$] : 00100010 : [imm$_{LSB}$] : [imm$_{MSB}$]
T State	20

Register	Bit
IX	11011101
IY	11111101

LD SP,HL

Operasi	Menyimpan isi HL ke dalam SP
Op Code	11111001
T State	6

LD SP,reg$_{index}$

Operasi	Menyimpan isi index register reg$_{index}$ ke dalam SP.
Op Code	[reg$_{index}$] : 11111001
T State	10

Register	Bit
IX	11011101
IY	11111101

d. Instruksi LDD

LDD

Operasi	Transfer data byte dari lokasi memori yang ditunjuk oleh HL ke dalammemori yang ditunjuk oleh DE. Kemudian HL, DE, dan BC dekremen.
Op Code	11101101 : 10101000
Flag	S Z C tidak berubah H N di-reset P/V di-reset jika BC menjadi nol.
T State	16

LDDR

Operasi	Transfer data byte dari lokasi memori yang ditunjuk oleh HL ke dalammemori yang ditunjuk oleh DE. Kemudian HL, DE, dan BC dekremen. Jika BC tidak nol, maka operasi ini diulang. Interupsi dapat di-trigger selama waktu instruksi ini diproses
Op Code	11101101 : 10111000
Flag	S Z C tidak berubah H N P/V di-reset
T State	Jika BC != 0: 21 Jika BC == 0: 16

LDI

Operasi	Transfer data byte dari lokasi memori yang ditunjuk oleh HL ke dalammemori yang ditunjuk oleh DE. Kemudian HL, DE, dan BC inkremen.

Op Code	11101101 : 10100000
Flag	S Z C tidak berubah H N di-reset P/V di-reset jika BC menjadi nol.
T State	16

LDIR

Operasi	Transfer data byte dari lokasi memori yang ditunjuk oleh HL ke dalammemori yang ditunjuk oleh DE. Kemudian HL, DE, dan BC dekremen . n HL dan DE are incremented dan BC is dekremen. Jika BC tidak sama dengan nol, operasi ini diulang. Interupsi dapat di-trigger selama waktu instruksi ini diproses.
Op Code	11101101 : 10110000
Flag	S Z C tidak berubah H N P/V di-reset
T State	Jika BC != 0: 21 Jika BC == 0: 16

e. Instruksi POP

POP reg$_{16}$

Operasi	Isi lokasi memori yang ditunjuk SP disimpan ke reg$_{LSB}$ dan SP inkremen. Isi lokasi memori yang ditunjuk SP disimpan ke reg$_{LSB}$ dan SP inkremen lagi.

<table>
<tr><td>Op Code</td><td>11[reg16]0001 <table><tr><th>Register</th><th>Bit</th></tr><tr><td>BC</td><td>00</td></tr><tr><td>DE</td><td>01</td></tr><tr><td>HL</td><td>10</td></tr><tr><td>AF</td><td>11</td></tr></table></td></tr>
<tr><td>T State</td><td>10</td></tr>
</table>

POP reg_{index}

<table>
<tr><td>Operasi</td><td>memory locatipada ditunjuk oleh SP is stored into reg_{LSB} dan SP inkremen. memory locatipada ditunjuk oleh SP is stored into reg_{MSB} dan SP inkremen again</td></tr>
<tr><td>Op Code</td><td>[reg_{index}] : 11100001 <table><tr><th>Register</th><th>Bit</th></tr><tr><td>IX</td><td>11011101</td></tr><tr><td>IY</td><td>11111101</td></tr></table></td></tr>
<tr><td>T State</td><td>14</td></tr>
</table>

f. Instruksi PUSH

PUSH reg_{16}

<table>
<tr><td>Operasi</td><td>SP dekremen dan reg_{MSB} disimpan kelokasi memori yang ditunjuk oleh SP. Kemudian SP dekremen lagi dan reg_{LSB} disimpan kelokasi memori yang ditunjuk oleh SP.</td></tr>
<tr><td>Op Code</td><td>11[reg_{16}]0101 <table><tr><th>Register</th><th>Bit</th></tr><tr><td>BC</td><td>00</td></tr><tr><td>DE</td><td>01</td></tr></table></td></tr>
</table>

	HL 10 AF 11
T State	11

PUSH reg_{index}

Operasi	SP dekremen dan reg_{MSB} disimpan kelokasi memori yang ditunjuk oleh SP. Kemudian SP dekremen lagi dan reg_{LSB} disimpan kelokasi memori yang ditunjuk oleh SP.
Op Code	[reg_{index}] : 11100101 **Register** / **Bit** IX / 11011101 IY / 11111101
T State	15

2. Instruksi Arithmatika Z80

CPU Z80 dalam melaksanakan Instruksi arithmetik 8 bit dan operasi-operasi logiknya selalu menggunakan register A sebagai akumulator, sedangkan register-register B, C, D, E, H, dan L difungsikan sebagai operan yang mendukung pelaksanaan instruksi tersebut. Adapun instruksi tersebut meliputi: ADC, ADD, CP, CPD, CPDR, CPI, CPIR, CPL, DAA, DEC, INC, NEG, SBC, SUB.

ADC A,reg_8

Operasi	reg_8 dan carry flag dijumlahkan dengan A.
Op Code	10001[reg_8] **Register** / **Bit** A / 111 B / 000 C / 001

<table>
<tr><td></td><td><table><tr><td>D</td><td>010</td></tr><tr><td>E</td><td>011</td></tr><tr><td>H</td><td>100</td></tr><tr><td>L</td><td>101</td></tr><tr><td>(HL)</td><td>110</td></tr></table></td></tr>
<tr><td>Flag</td><td>S Z H C berubah sesuai yang diinginkan (ditentukan)
P/V terdapat overflow
N di-reset</td></tr>
<tr><td>T State</td><td>4 atau 7 (HL)</td></tr>
</table>

ADC A,imm$_8$

Operasi	nilai immediate dan carry flag dijumlahkan dengan A.
Op Code	11001110 : [imm$_8$]
Flag	S Z H C berubah sesuai yang diinginkan (ditentukan) P/V terdapat overflow N di-reset
T State	7

ADC A,(reg$_{index}$ + ofs$_8$)

<table>
<tr><td>Operasi</td><td>Isi memori yang ditunjuk oleh index register reg$_{index}$ ditambah ofs$_8$, dan carry flag, dijumlahkan dengan A.</td></tr>
<tr><td>Op Code</td><td>[reg$_{index}$] : 10001110 : [ofs$_8$]
<table><tr><td>Register</td><td>Bit</td></tr><tr><td>IX</td><td>11011101</td></tr><tr><td>IY</td><td>11111101</td></tr></table></td></tr>
<tr><td>Flag</td><td>S Z H C berubah sesuai yang diinginkan (ditentukan)
P/V terdapat overflow
N di-reset</td></tr>
</table>

T State	19

ADC HL,reg$_{16}$

<table>
<tr><td>Operasi</td><td>Nilai dari reg$_{16}$ dan carry flag dijumlahkan dengan HL.</td></tr>
<tr><td>Op Code</td><td>11101101 : 01[reg$_{16}$]1010
<table><tr><th>Register</th><th>Bit</th></tr><tr><td>BC</td><td>00</td></tr><tr><td>DE</td><td>01</td></tr><tr><td>HL</td><td>10</td></tr><tr><td>SP</td><td>11</td></tr></table></td></tr>
<tr><td>Flag</td><td>S Z C berubah sesuai yang diinginkan (ditentukan)
P/V terdapat overflow
H di-set jika terjadi carry out pada bit 11
N di-reset</td></tr>
<tr><td>T State</td><td>15</td></tr>
</table>

ADD A,reg$_8$

<table>
<tr><td>Operasi</td><td>Menambah reg$_8$ ke A.</td></tr>
<tr><td>Op Code</td><td>10000[reg$_8$]
<table><tr><th>Register</th><th>Bit</th></tr><tr><td>A</td><td>111</td></tr><tr><td>B</td><td>000</td></tr><tr><td>C</td><td>001</td></tr><tr><td>D</td><td>010</td></tr><tr><td>E</td><td>011</td></tr><tr><td>H</td><td>100</td></tr><tr><td>L</td><td>101</td></tr></table></td></tr>
</table>

	(HL) | 110
Flag	S Z H C berubah sesuai yang diinginkan (ditentukan) P/V terdapat overflow, N di-clear (dinolkan)
T State	4 atau 7 (HL)

ADD A,imm$_8$

Operasi	Nilai immediate ditambahkan dengan A.
Op Code	11000110 : [imm$_8$]
Flag	S Z H C berubah sesuai yang diinginkan (ditentukan) P/V terdapat overflow N di-clear (dinolkan)
T State	7

ADD A,(reg$_{index}$ + ofs$_8$)

Operasi	Menambah isi memori pada lokasi ditunjuk oleh reg$_{index}$ ditambah ofs$_8$ dengan A.
Op Code	[reg$_{index}$] : 10000110 : [ofs$_8$] **Register** | **Bit** IX | 11011101 IY | 11111101
Flag	S Z H C berubah sesuai yang diinginkan (ditentukan) P/V terdapat overflow, N di-clear (dinolkan)
T State	19

ADD HL,reg$_{16}$

<table>
<tr><td>Operasi</td><td>Nilai darireg$_{16}$ ditambahkan dengan HL.</td></tr>
<tr><td>Op Code</td><td>00[reg$_{16}$]1001
<table>
<tr><th>Register</th><th>Bit</th></tr>
<tr><td>BC</td><td>00</td></tr>
<tr><td>DE</td><td>01</td></tr>
<tr><td>HL</td><td>10</td></tr>
<tr><td>SP</td><td>11</td></tr>
</table></td></tr>
<tr><td>Flag</td><td>S Z P/V tidak berubah
C berubah sesuai yang diinginkan (ditentukan)
H di-set jika terjadi carry out pada bit 11, N di-reset</td></tr>
<tr><td>T State</td><td>11</td></tr>
</table>

ADD IX,reg$_{16}$

<table>
<tr><td>Operasi</td><td>Nilai darireg$_{16}$ ditambahkan dengan IX.</td></tr>
<tr><td>Op Code</td><td>11011101 : 00[reg$_{16}$]1001
<table>
<tr><th>Register</th><th>Bit</th></tr>
<tr><td>BC</td><td>00</td></tr>
<tr><td>DE</td><td>01</td></tr>
<tr><td>IX</td><td>10</td></tr>
<tr><td>SP</td><td>11</td></tr>
</table></td></tr>
<tr><td>Flag</td><td>S Z P/V tidak berubah
C berubah sesuai yang diinginkan (ditentukan)
H di-set jika terjadi carry out pada bit 11, N di-reset</td></tr>
<tr><td>T State</td><td>15</td></tr>
</table>



ADD IY,reg_{16}

<table>
<tr><td>Operasi</td><td>Nilai dari reg_{16} ditambahkan dengan IY.</td></tr>
<tr><td>Op Code</td><td>11111101 : 00[reg_{16}]1001
<table>
<tr><th>Register</th><th>Bit</th></tr>
<tr><td>BC</td><td>00</td></tr>
<tr><td>DE</td><td>01</td></tr>
<tr><td>IY</td><td>10</td></tr>
<tr><td>SP</td><td>11</td></tr>
</table></td></tr>
<tr><td>Flag</td><td>S Z P/V tidak berubah
C berubah sesuai yang diinginkan (ditentukan)
H di-set jika terjadi carry out pada bit 11
N di-reset</td></tr>
<tr><td>T State</td><td>15</td></tr>
</table>

CP reg_8

<table>
<tr><td>Operasi</td><td>Mengurangkan reg_8 dari A dan merubah Flag terkait dengan hasil. A tidak dimodifikasi</td></tr>
<tr><td>Op Code</td><td>10111[reg_8]
<table>
<tr><th>Register</th><th>Bit</th></tr>
<tr><td>A</td><td>111</td></tr>
<tr><td>B</td><td>000</td></tr>
<tr><td>C</td><td>001</td></tr>
<tr><td>D</td><td>010</td></tr>
<tr><td>E</td><td>011</td></tr>
<tr><td>H</td><td>100</td></tr>
<tr><td>L</td><td>101</td></tr>
<tr><td>(HL)</td><td>110</td></tr>
</table></td></tr>
<tr><td>Flag</td><td>S Z H C berubah sesuai yang diinginkan (ditentukan)</td></tr>
</table>

	P/V terdapat overflow N di-set
T State	4 atau 7 (HL)

CP imm_8

Operasi	Mengurangkan nilai immediate dari A dan perubahan Flag. A tidak berubah.
Op Code	11111110 : [imm_8]
Flag	S Z H C berubah sesuai yang diinginkan (ditentukan) P/V terdapat overflow N di-set
T State	7

CP (reg_{index} + ofs_8)

Operasi	Mengurangkan isi memori ditunjuk oleh reg_{index} ditambah ofs_8 dari A dan merubah Flag terkait dengan hasil. A tidak dimodifikasi
Op Code	[reg_{index}] : 10111110 : [ofs_8] **Register** / **Bit**: IX 11011101; IY 11111101
Flag	S Z H C berubah sesuai yang diinginkan (ditentukan) P/V terdapat overflow N di-set
T State	19

Register	**Bit**
IX	11011101
IY	11111101

CPD

Operasi	Membandingkan isi memori pada lokasi ditunjuk oleh HL dengan A. HL dan BC dekremen.
Op Code	11101101 : 10101001
Flag	S Z H berubah sesuai yang diinginkan (ditentukan) P/V di-reset jika BC menjadi nol N di-set C tidak berubah
T State	16

CPDR

Operasi	Membandingkan isi memori pada lokasi ditunjuk oleh HL dengan A. HL dan BC dekremen. Jika BC is tidak nol dan Z tidak di-set, operasi ini diulang. Interupsi dapat di-trigger sementara instruksi diproses.
Op Code	11101101 : 10111001
Flag	S Z H berubah sesuai yang diinginkan (ditentukan) P/V di-reset jika BC menjadi nol N di-set C tidak berubah
T State	Jika BC != 0 dan Z di-reset: 21 Jika BC == 0 atau Z di-set: 16

CPI

Operasi	Membandingkan isi memori pada lokasi ditunjuk oleh HL dengan A. HL inkremen dan BC dekremen.
Op Code	11101101 : 10100001
Flag	S Z H berubah sesuai yang diinginkan (ditentukan)

	P/V di-reset jika BC menjadi nol N di-set C tidak berubah
T State	16

CPIR

Operasi	Membandingkan isi memori pada lokasi ditunjuk oleh HL dengan A. HL inkremen dan BC dekremen. Jika BC tidak nol dan Z tidak di-set, operasi ini diulang. Interupsi dapat di-trigger selama instruksi diproses.
Op Code	11101101 : 10110001
Flag	S Z H berubah sesuai yang diinginkan (ditentukan) P/V di-reset jika BC menjadi nol N di-set C tidak berubah
T State	Jika BC != 0 dan Z di-reset: 21 Jika BC == 0 atu Z di-set: 16

CPL

Operasi	Isi dari A dibalikan (komplemen satu).
Op Code	00101111
Flag	S Z P/V C tidak berubah H N are set
T State	4

DAA

Operasi	Mengatur A untuk Operasi penjumlah dan pengurangan BCD.

	N Flag	C Flag	Bit 7-4	H Flag	Bit 3-0	Nilai yang ditambahkan	Output C Flag
	0	0	9-0	0	0-9	00	1
		0	0-8	0	A-F	06	1
		0	0-9	1	0-3	06	1
		0	A-F	0	0-9	60	0
		0	9-F	0	A-F	66	0
		0	A-F	1	0-3	66	0
		1	0-2	0	0-9	60	0
		1	0-2	0	A-F	66	0
		1	0-3	1	0-3	66	0
	1	0	0-9	0	0-9	00	0
		0	0-8	1	6-F	FA	0
		1	7-F	0	0-9	A0	1
		1	6-7	1	6-F	9A	1
Op Code	00100111						
Flag	S Z berubah sesuai yang diinginkan (ditentukan) P/V merupakan paritas N tidak berubah Lihat instruksi untuk H C						
T State	4						

DEC reg_8

Operasi	Mengurangkan satu pada reg_8.
Op Code	00[reg_8]101

Register	Bit
A	111
B	000
C	001

	<table><tr><td>D</td><td>010</td></tr><tr><td>E</td><td>011</td></tr><tr><td>H</td><td>100</td></tr><tr><td>L</td><td>101</td></tr><tr><td>(HL)</td><td>110</td></tr></table>
Flag	S Z H berubah sesuai yang diinginkan (ditentukan) P/V di-set jika operan $80 sebelum Operasi N di-set C tidak berubah
T State	4 atau 11 (HL)

DEC (reg_{index} + ofs_8)

Operasi	Mengurangkan isi memori pada lokasi yang ditunjuk oleh reg_{index} ditambah ofs_8.
Op Code	[reg_{index}] : 00110101 : [ofs_8] <table><tr><th>Register</th><th>Bit</th></tr><tr><td>IX</td><td>11011101</td></tr><tr><td>IY</td><td>11111101</td></tr></table>
Flag	S Z H berubah sesuai yang diinginkan (ditentukan) P/V di-set jika operan $80 sebelum Operasi N di-set C tidak berubah
T State	23

DEC reg_{16}

Operasi	Mengurangkan padae dari reg_{16}.
Op Code	00[reg_{16}]1011



<table>
<tr><td></td><td><table><tr><th>Register</th><th>Bit</th></tr><tr><td>BC</td><td>00</td></tr><tr><td>DE</td><td>01</td></tr><tr><td>HL</td><td>10</td></tr><tr><td>SP</td><td>11</td></tr></table></td></tr>
<tr><td>T State</td><td>6</td></tr>
</table>

DEC reg_{index}

<table>
<tr><td>Operasi</td><td>Mengurangkan dengan satu pada reg_{index}.</td></tr>
<tr><td>Op Code</td><td>[reg_{index}] : 00101011<table><tr><th>Register</th><th>Bit</th></tr><tr><td>IX</td><td>11011101</td></tr><tr><td>IY</td><td>11111101</td></tr></table></td></tr>
<tr><td>T State</td><td>10</td></tr>
</table>

INC reg_8

<table>
<tr><td>Operasi</td><td>Menambah satu ke reg_8.</td></tr>
<tr><td>Op Code</td><td>00[reg_8]100<table><tr><th>Register</th><th>Bit</th></tr><tr><td>A</td><td>111</td></tr><tr><td>B</td><td>000</td></tr><tr><td>C</td><td>001</td></tr><tr><td>D</td><td>010</td></tr><tr><td>E</td><td>011</td></tr><tr><td>H</td><td>100</td></tr><tr><td>L</td><td>101</td></tr><tr><td>(HL)</td><td>110</td></tr></table></td></tr>
</table>

<table>
<tr><td>Flag</td><td>S Z H berubah sesuai yang diinginkan (ditentukan)
P/V di-set jika operan $7F sebelum Operasi
C tidak berubah</td></tr>
<tr><td>T State</td><td>4 atau 11 (HL)</td></tr>
</table>

INC (reg$_{index}$ + ofs$_{8}$)

<table>
<tr><td>Operasi</td><td>Menambah satu ke lokasi memori yang ditunjuk oleh reg$_{index}$ ditambah ofs$_{8}$.</td></tr>
<tr><td>Op Code</td><td>[reg$_{index}$] : 00110100 : [ofs$_{8}$]
<table>
<tr><td>Register</td><td>Bit</td></tr>
<tr><td>IX</td><td>11011101</td></tr>
<tr><td>IY</td><td>11111101</td></tr>
</table></td></tr>
<tr><td>Flag</td><td>S Z H berubah sesuai yang diinginkan (ditentukan)
P/V di-set jika operan $7F sebelum Operasi
C tidak berubah</td></tr>
<tr><td>T State</td><td>23</td></tr>
</table>

INC reg$_{16}$

<table>
<tr><td>Operasi</td><td>Menambah satu ke reg$_{16}$.</td></tr>
<tr><td>Op Code</td><td>00[reg$_{16}$]0011
<table>
<tr><td>Register</td><td>Bit</td></tr>
<tr><td>BC</td><td>00</td></tr>
<tr><td>DE</td><td>01</td></tr>
<tr><td>HL</td><td>10</td></tr>
<tr><td>SP</td><td>11</td></tr>
</table></td></tr>
<tr><td>T State</td><td>6</td></tr>
</table>

INC reg_{index}

<table>
<tr><td>Operasi</td><td>Menambah satu ke reg_{index}.</td></tr>
<tr><td>Op Code</td><td>[reg_{index}] : 00100011
<table><tr><td>Register</td><td>Bit</td></tr><tr><td>IX</td><td>11011101</td></tr><tr><td>IY</td><td>11111101</td></tr></table></td></tr>
<tr><td>T State</td><td>10</td></tr>
</table>

NEG

Operasi	Isi dari A dijadikan komplemen dua (negated). Operasi sama seperti pengurangan A dari dari.
Op Code	11101101 : 01000100
Flag	S Z H berubah sesuai yang diinginkan (ditentukan) P/V di-set jika A $80 sebelum Operasi N di-set C di-set jika A tidak $00 sebelum Operasi
T State	8

SBC A,reg_8

<table>
<tr><td>Operasi</td><td>Mengurangkan reg_8 dan carry flag dari A.</td></tr>
<tr><td>Op Code</td><td>10011[reg_8]
<table><tr><td>Register</td><td>Bit</td></tr><tr><td>A</td><td>111</td></tr><tr><td>B</td><td>000</td></tr><tr><td>C</td><td>001</td></tr><tr><td>D</td><td>010</td></tr><tr><td>E</td><td>011</td></tr><tr><td>H</td><td>100</td></tr></table></td></tr>
</table>

	L: 101 (HL): 110
Flag	S Z H C berubah sesuai yang diinginkan (ditentukan) P/V terdapat overflow N di-set
T State	4 atau 7 (HL)

SBC A,imm$_8$

Operasi	Mengurangkan nilai immediate dan carry flag dari A.
Op Code	11011110 : [imm$_8$]
Flag	S Z H C berubah sesuai yang diinginkan (ditentukan) P/V terdapat overflow N di-set
T State	7

SBC A,(reg$_{index}$ + ofs$_8$)

Operasi	Mengurangkan isi lokasi memori yang ditunjuk oleh reg$_{index}$ ditambah ofs$_8$, dan carry flag dari A
Op Code	[reg$_{16}$] : 10011110 : [ofs$_8$] **Register** / **Bit** IX: 11011101 IY: 11111101
Flag	S Z H C berubah sesuai yang diinginkan (ditentukan) P/V terdapat overflow N di-set
T State	19



SBC HL,reg$_{16}$

<table>
<tr><td>Operasi</td><td>Mengurangkan reg$_{16}$ dan carry flag dari HL.</td></tr>
<tr><td>Op Code</td><td>11101101 : 01[reg$_{16}$]0010
<table>
<tr><th>Register</th><th>Bit</th></tr>
<tr><td>BC</td><td>00</td></tr>
<tr><td>DE</td><td>01</td></tr>
<tr><td>HL</td><td>10</td></tr>
<tr><td>SP</td><td>11</td></tr>
</table></td></tr>
<tr><td>Flag</td><td>S Z C berubah sesuai yang diinginkan (ditentukan)
H di-set jika a borrow dari bit 12
P/V terdapat overflow
N di-set</td></tr>
<tr><td>T State</td><td>15</td></tr>
</table>

SUB A,reg$_8$

<table>
<tr><td>Operasi</td><td>Mengurangkan reg$_8$ dari A.</td></tr>
<tr><td>Op Code</td><td>10010[reg$_8$]
<table>
<tr><th>Register</th><th>Bit</th></tr>
<tr><td>A</td><td>111</td></tr>
<tr><td>B</td><td>000</td></tr>
<tr><td>C</td><td>001</td></tr>
<tr><td>D</td><td>010</td></tr>
<tr><td>E</td><td>011</td></tr>
<tr><td>H</td><td>100</td></tr>
<tr><td>L</td><td>101</td></tr>
<tr><td>(HL)</td><td>110</td></tr>
</table></td></tr>
<tr><td>Flag</td><td>S Z H C berubah sesuai yang diinginkan (ditentukan)
P/V terdapat overflow</td></tr>
</table>

	N di-set
T State	4 atau 7 (HL)

SUB A,imm$_8$

Operasi	Mengurangkan nilai immediate dari A.
Op Code	11010110 : [imm$_8$]
Flag	S Z H C berubah sesuai yang diinginkan (ditentukan) P/V terdapat overflow N di-set
T State	7

SUB A,(reg$_{index}$ + ofs$_8$)

<table>
<tr><td>Operasi</td><td>Mengurangkan isi lokasi memori yang ditunjuk oleh reg_{index} ditambah ofs₈ dari A</td></tr>
<tr><td>Op Code</td><td>[reg₁₆] : 10010110 : [ofs₈]
<table>
<tr><td>Register</td><td>Bit</td></tr>
<tr><td>IX</td><td>11011101</td></tr>
<tr><td>IY</td><td>11111101</td></tr>
</table></td></tr>
<tr><td>Flag</td><td>S Z H C berubah sesuai yang diinginkan (ditentukan)
P/V terdapat overflow
N di-set</td></tr>
<tr><td>T State</td><td>19</td></tr>
</table>

3. Instruksi Bit Z80

Operasi Bit pada Z80 pada prinsipnya merupakan pelaksanaan instruksi logika yang menggunakan register A sebagai akumulator dan ditunjang oleh register dasar lainnya. Adapun instruksi Bit ini meliputi AND, BIT, CCF, OR, RES, SCF, SET, XOR untuk lebih jelasnya kita ikuti penjelasan setiap instruksi tersebut.

AND reg_8

<table>
<tr><td>Operasi</td><td>Bitwise AND pada A dengan reg_8.</td></tr>
<tr><td>Op Code</td><td>10100[reg_8]
<table>
<tr><th>Register</th><th>Bit</th></tr>
<tr><td>A</td><td>111</td></tr>
<tr><td>B</td><td>000</td></tr>
<tr><td>C</td><td>001</td></tr>
<tr><td>D</td><td>010</td></tr>
<tr><td>E</td><td>011</td></tr>
<tr><td>H</td><td>100</td></tr>
<tr><td>L</td><td>101</td></tr>
<tr><td>(HL)</td><td>110</td></tr>
</table></td></tr>
<tr><td>Flag</td><td>S Z berubah sesuai yang diinginkan (ditentukan)
H di-set
P/V merupakan paritas
N C di-reset</td></tr>
<tr><td>T State</td><td>4 atau 7 (HL)</td></tr>
</table>

AND imm_8

Operasi	Bitwise AND pada A dengan imm_8.
Op Code	11100110 : [imm_8]
Flag	S Z berubah sesuai yang diinginkan (ditentukan) H di-set P/V merupakan paritas N C di-reset
T State	7

AND (reg$_{index}$ + ofs$_8$)

<table>
<tr><td>Operasi</td><td>Bitwise AND pada A dengan data pada lokasi memori ditunjuk oleh reg$_{index}$ ditambah ofs$_8$.</td></tr>
<tr><td>Op Code</td><td>[reg$_{index}$] : 10100110 [ofs$_8$]
<table><tr><th>Register</th><th>Bit</th></tr><tr><td>IX</td><td>11011101</td></tr><tr><td>IY</td><td>11111101</td></tr></table></td></tr>
<tr><td>Flag</td><td>S Z berubah sesuai yang diinginkan (ditentukan)
H di-set, P/V merupakan paritas, N C di-reset</td></tr>
<tr><td>T State</td><td>19</td></tr>
</table>

BIT imm$_3$,reg$_8$

<table>
<tr><td>Operasi</td><td>Tes bit imm$_3$ dari reg$_8$.</td></tr>
<tr><td>Op Code</td><td>11001011 : 01[imm$_3$][reg$_8$]
<table><tr><th>Register</th><th>Bit</th></tr><tr><td>A</td><td>111</td></tr><tr><td>B</td><td>000</td></tr><tr><td>C</td><td>001</td></tr><tr><td>D</td><td>010</td></tr><tr><td>E</td><td>011</td></tr><tr><td>H</td><td>100</td></tr><tr><td>L</td><td>101</td></tr><tr><td>(HL)</td><td>110</td></tr></table></td></tr>
<tr><td>Flag</td><td>S P/V are scrambled
Z berubah sesuai yang diinginkan (ditentukan)
H di-set, N di-clear (dinolkan)
C tidak berubah</td></tr>
<tr><td>T State</td><td>8 atau 12 (HL)</td></tr>
</table>

BIT imm_3,(reg_{index} + ofs_8)

<table>
<tr><td>Operasi</td><td>Tes bit imm_3 data pada lokasi memori ditunjuk oleh reg_{index} ditambah ofs_8.</td></tr>
<tr><td>Op Code</td><td>[reg_{index}] : 11001011 : [ofs_8] : 01[$imm_3$110
<table>
<tr><td>Register</td><td>Bit</td></tr>
<tr><td>IX</td><td>11011101</td></tr>
<tr><td>IY</td><td>11111101</td></tr>
</table></td></tr>
<tr><td>Flag</td><td>S P/V are scrambled
Z berubah sesuai yang diinginkan (ditentukan)
H di-set, N di-clear (dinolkan)
C tidak berubah</td></tr>
<tr><td>T State</td><td>23</td></tr>
</table>

CCF

Operasi	Pembalikan Nilai dari carry flag.
Op Code	00111111
Flag	S Z P/V tidak berubah H is carry beuntuke Operasi, N di-reset, Lihat instruksi untuk C
T State	4

OR reg_8

<table>
<tr><td>Operasi</td><td>Bitwise OR pada A dengan reg_8.</td></tr>
<tr><td>Op Code</td><td>10110[reg_8]
<table>
<tr><td>Register</td><td>Bit</td></tr>
<tr><td>A</td><td>111</td></tr>
<tr><td>B</td><td>000</td></tr>
<tr><td>C</td><td>001</td></tr>
</table></td></tr>
</table>

	D 010 E 011 H 100 L 101 (HL) 110
Flag	S Z berubah sesuai yang diinginkan (ditentukan) P/V merupakan paritas, H N C di-reset
T State	4 atau 7 (HL)

OR imm_8

Operasi	Bitwise OR pada A denganimm$_8$.
Op Code	11110110 : [imm$_8$]
Flag	S Z berubah sesuai yang diinginkan (ditentukan) P/V merupakan paritas, H N C di-reset
T State	7

OR (reg_{index} + ofs_8)

Operasi	Bitwise OR pada A dengan data pada lokasi memori ditunjuk oleh reg$_{index}$ ditambah ofs$_8$.
Op Code	[reg$_{index}$] : 10110110 [ofs$_8$] **Register** **Bit** IX 11011101 IY 11111101
Flag	S Z berubah sesuai yang diinginkan (ditentukan) P/V merupakan paritas, H N C di-reset
T State	19

RES imm_3,reg_8

<table>
<tr><td>Operasi</td><td>Reset bit imm_3 dari reg_8.</td></tr>
<tr><td>Op Code</td><td>11001011 : 10[imm_3][reg_8]
<table>
<tr><th>Register</th><th>Bit</th></tr>
<tr><td>A</td><td>111</td></tr>
<tr><td>B</td><td>000</td></tr>
<tr><td>C</td><td>001</td></tr>
<tr><td>D</td><td>010</td></tr>
<tr><td>E</td><td>011</td></tr>
<tr><td>H</td><td>100</td></tr>
<tr><td>L</td><td>101</td></tr>
<tr><td>(HL)</td><td>110</td></tr>
</table></td></tr>
<tr><td>T State</td><td>8 atau 15 (HL)</td></tr>
</table>

RES imm_3,(reg_{index} + ofs_8)

Operasi	Reset bit imm_3 dari Nilai pada lokasi memori ditunjuk oleh reg_{index} ditambah ofs_8.
Op Code	[reg_{index}] : 11001011 : [ofs_8] : 10[imm_3]110
Flag	Tidak berpengaruh
T State	23

SCF

Operasi	Sets carry flag.
Op Code	00110111
Flag	S Z P/V tidak berubah H N di-reset C di-set

T State	4

SET imm_3,reg_8

<table>
<tr><td>Operasi</td><td>Sets bit imm_3 pada reg_8.</td></tr>
<tr><td>Op Code</td><td>11001011 : 11[imm_3][reg_8]
<table>
<tr><th>Register</th><th>Bit</th></tr>
<tr><td>A</td><td>111</td></tr>
<tr><td>B</td><td>000</td></tr>
<tr><td>C</td><td>001</td></tr>
<tr><td>D</td><td>010</td></tr>
<tr><td>E</td><td>011</td></tr>
<tr><td>H</td><td>100</td></tr>
<tr><td>L</td><td>101</td></tr>
<tr><td>(HL)</td><td>110</td></tr>
</table></td></tr>
<tr><td>T State</td><td>8 atau 15 (HL)</td></tr>
</table>

SET imm_3,(reg_{index} + ofs_8)

Operasi	Sets bit imm_3 dari Nilai pada lokasi memori ditunjuk oleh reg_{index} ditambah ofs_8.
Op Code	[reg_{index}] : 11001011 : [ofs_8] : 11[imm_3]110
Flag	Tidak berpengaruh
T State	23

XOR reg_8

Operasi	Bitwise XOR pada A dengan reg_8.
Op Code	10101[reg_8]
Flag	S Z berubah sesuai yang diinginkan (ditentukan) H di-set P/V merupakan paritas N C di-reset
T State	4 atau 7 (HL)

Register	**Bit**
A	111
B	000
C	001
D	010
E	011
H	100
L	101
(HL)	110

XOR imm_8

Operasi	Bitwise XOR pada A denganimm$_8$.
Op Code	11101110 : [imm_8]
Flag	S Z berubah sesuai yang diinginkan (ditentukan) H di-set P/V merupakan paritas N C di-reset
T State	7

XOR (reg_{index} + ofs_8)

Operasi	Bitwise XOR pada A dengan data pada lokasi memori ditunjuk oleh reg_{index} ditambah ofs_8.
Op Code	[reg_{index}] : 10101110 [ofs_8] <table><tr><th>Register</th><th>Bit</th></tr><tr><td>IX</td><td>11011101</td></tr><tr><td>IY</td><td>11111101</td></tr></table>
Flag	S Z berubah sesuai yang diinginkan (ditentukan) H di-set P/V merupakan paritas N C di-reset
T State	19

4. Instruksi Geser/Putar Z80

Instruksi geser dan putar merupakan instruksi manipulasi bit yang ada pada register-register, seperti telah dijelaskan pada bab sebelumnya bahwa register dapat digeser ke kiri atau ke kanan dan dapat pula isi register diputar sehingga bit ke 7 menempati posisi bit ke 0.

Sedangkan instruksi geser dan putar pada mikroprosesor Z80 meliputi **RL, RLA, RLC, RLCA, RLD, RR, RRA, RRC, RRCA, RRD, SLA, SRA, SRL.**

a. Instruksi RL

RL reg_8

Operasi	Isi dari reg_8 diputar ke kiri posisi satu bit. Isi dari bit 7 disalin ke carry flag dan isi carry flag sebelumnya disalin ke bit 0.
Op Code	11001011 : 00010[reg_8] <table><tr><th>Register</th><th>Bit</th></tr><tr><td>A</td><td>111</td></tr><tr><td>B</td><td>000</td></tr></table>

C	001
D	010
E	011
H	100
L	101
(HL)	110

Flag	S Z berubah sesuai yang diinginkan (ditentukan) H N di-reset, P/V merupakan paritas, Lihat instruksi untuk C
T State	8 atau 15 (HL)

RL (reg_{index} + ofs_8)

Operasi	Isi memori pada lokasi ditunjuk oleh reg_{index} ditambah ofs_8 diputar ke kiri satu posisi bit. Isi dari bit 7 disalin ke carry flag dan isi carry flag sebelumnya disalin ke are bit 0.
Op Code	[reg_{index}] : 11001011 : [ofs_8] : 00010110

Register	Bit
IX	11011101
IY	11111101

Flag	S Z berubah sesuai yang diinginkan (ditentukan) H N di-reset, P/V merupakan paritas, Lihat instruksi untuk C
T State	23

b. Instruksi RLA

RLA

Operasi	Isi dari A diputar ke kiri satu posisi bit. Isi dari bit 7 disalin ke carry flag dan isi carry flag sebelumnya disalin ke bit 0.
Op Code	00010111
Flag	S Z P/V tidak berubah H N di-reset Lihat instruksi untuk C.
T State	4

c. Instruksi RLC

RLC reg_8

Operasi	Isi dari reg_8 digeser ke kiri satu posisi bit. Isi dari bit 7 disalin ke carry flag dan ke bit 0.
Op Code	11001011 : 00000[reg_8] **Register** / **Bit**: A 111 B 000 C 001 D 010 E 011 H 100 L 101 (HL) 110
Flag	S Z berubah sesuai yang diinginkan (ditentukan) H N di-reset P/V merupakan paritas Lihat instruksi untuk C

T State	8 atau 15 (HL)

RLC (reg_{index} + ofs_8)

Operasi	Data pada lokasi memori ditunjuk oleh reg_{index} ditambah ofs_8 diputar ke kiri, dan isi dari bit 7 di salin dan diletakan pada carry flag dan bit 0.
Op Code	[reg_{index}] : 11001011 : [ofs_8] : 00000110 **Register** / **Bit** IX / 11011101 IY / 11111101
Flag	S Z berubah sesuai yang diinginkan (ditentukan) H N di-reset P/V merupakan paritas Lihat instruksi untuk C
T State	23

d. Instruksi RLCA

RLCA

Operasi	Isi dari A diputar posisinya 1 bit ke kiri. Isi dari bit ke 7 disalin ke carry flag dan bit 0.
Op Code	00000111
Flag	S Z P/V tidak berubah H N di-reset Lihat instruksi untuk C
T State	4

e. Instruksi RLD

RLD

Operasi	Isi dari low-order nibble dari (HL) disalin ke high-order nibble dari (HL). Isi sebelumnya disalin ke low-order nibble dari A. Isi sebelumnya disalin ke low-order nibble dari (HL).
Op Code	11101101 : 01101111
Flag	Flag refer to state dari A S Z berubah sesuai yang diinginkan (ditentukan) H N di-reset P/V merupakan paritas C tidak terpengaruh
T State	18

f. Instruksi RR

RR reg$_8$

Operasi	Isi dari reg$_8$ diputar posisinya ke kanan 1 bit. Isi bit 0 disalin ke carry flag dan isi sebelumnya disalin ke bit 7.
Op Code	11001011 : 00011[reg$_8$]

Register	Bit
A	111
B	000
C	001
D	010
E	011

	H 100 L 101 (HL) 110
Flag	S Z berubah sesuai yang diinginkan (ditentukan) H N di-reset P/V merupakan paritas Lihat instruksi untuk C
T State	8 atau 15 (HL)

RR (reg_{index} + ofs_8)

Operasi	Isi reg_8 diputar posisinya ke kanan 1 bit. Isi bit 0 disalin ke carry flag dan isi carry flag sebelumnya disalin ke bit 7.
tOp Code	[reg_{index}] : 11001011 : [ofs_8] : 00011110 **Register** / **Bit** IX 11011101 IY 11111101
Flag	S Z berubah sesuai yang diinginkan (ditentukan) H N di-reset P/V merupakan paritas Lihat instruksi untuk C
T State	23

g. Instruksi RRA

RRA

Operasi	Isi dari A posisi satu bit diputar ke kanan. Isi dari bit 0 disalin ke carry flag dan isi carry flag sebelumnya disalin ke bit 7.
Op Code	00011111
Flag	S Z P/V tidak berubah H N di-reset Lihat instruksi untuk C
T State	4

h. Instruksi RRC

RRC reg_8

<table>
<tr><td>Operasi</td><td>Isi dari reg8 are diputar posisinya 1 bit ke kanan. Isi dari bit 0 disalin ke carry flag dan bit 7.</td></tr>
<tr><td>Op Code</td><td>11001011 : 00001[reg8]
<table>
<tr><th>Register</th><th>Bit</th></tr>
<tr><td>A</td><td>111</td></tr>
<tr><td>B</td><td>000</td></tr>
<tr><td>C</td><td>001</td></tr>
<tr><td>D</td><td>010</td></tr>
<tr><td>E</td><td>011</td></tr>
<tr><td>H</td><td>100</td></tr>
<tr><td>L</td><td>101</td></tr>
</table></td></tr>
</table>

	(HL) 110
Flag	S Z berubah sesuai yang diinginkan (ditentukan) H N di-reset P/V merupakan paritas Lihat instruksi untuk C
T State	8 atau 15 (HL)

RRC (reg_{index} + ofs_8)

Operasi	Isi memori pada lokasi yang ditunjukk oleh reg_{index} ditambah ofs_8 diputar posisinya 1 bit ke kanan. Isi bit 0 disalin ke carry flag dan bit 7.
Op Code	[reg_{index}] : 11001011 : [ofs_8] : 00001110 **Register** / **Bit** IX / 11011101 IY / 11111101
Flag	S Z berubah sesuai yang diinginkan (ditentukan) H N di-reset P/V merupakan paritas Lihat instruksi untuk C
T State	23

i. Instruksi RRCA

RRCA

Operasi	Isi dari A diputar posisinya 1 bit ke kanan. Isi dari bit 0 disalin ke carry flag dan bit 7.
Op Code	00001111
Flag	S Z P/V tidak berubah H N di-reset Lihat instruksi untuk C.
T State	4

j. Instruksi RRD

RRD

Operasi	Isi dari low-order nibble (HL) disalin ke low-order nibble A. Isi sebelum nya disalin ke high-order nibble (HL). Isi sebelumnya disalin ke low-order nibble (HL).
Op Code	11101101 : 01100111
Flag	Flag refer to state dari A S Z berubah sesuai yang diinginkan (ditentukan) H N di-reset P/V merupakan paritas Tidak berpengaruh pada C
T State	18

k. Instruksi SLA

SLA reg_8

<table>
<tr><td>Operasi</td><td>Isi dari reg_8 posisinya digeser ke kiri 1 bit, Isi bit 7 disalin ke carry flag dan isi dari zero dimasukan ke bit 0.</td></tr>
<tr><td>Op Code</td><td>11001011 : 00100[reg_8]
<table>
<tr><td>Register</td><td>Bit</td></tr>
<tr><td>A</td><td>111</td></tr>
<tr><td>B</td><td>000</td></tr>
<tr><td>C</td><td>001</td></tr>
<tr><td>D</td><td>010</td></tr>
<tr><td>E</td><td>011</td></tr>
<tr><td>H</td><td>100</td></tr>
<tr><td>L</td><td>101</td></tr>
<tr><td>(HL)</td><td>110</td></tr>
</table></td></tr>
<tr><td>Flag</td><td>S Z berubah sesuai yang diinginkan (ditentukan)
H N di-reset
P/V merupakan paritas
Lihat instruksi untuk C</td></tr>
<tr><td>T State</td><td>8 atau 15 (HL)</td></tr>
</table>

SLA (reg_{index} + ofs_8)

Operasi	Isi memori pada lokasi ditunjuk oleh reg_{index} ditambah ofs_8 posisinya digeser ke kiri 1 bit. Isi dari bit 7 disalin ke carry flag dan isi zero diletakan pada bit 0.

<table>
<tr><td>Op Code</td><td>[reg_{index}] : 11001011 : [ofs₈] : 00100110
<table>
<tr><th>Register</th><th>Bit</th></tr>
<tr><td>IX</td><td>11011101</td></tr>
<tr><td>IY</td><td>11111101</td></tr>
</table></td></tr>
<tr><td>Flag</td><td>S Z berubah sesuai yang diinginkan (ditentukan)
H N di-reset
P/V merupakan paritas
Lihat instruksi untuk C</td></tr>
<tr><td>T State</td><td>23</td></tr>
</table>

I. Instruksi SRA

SRA reg_8

<table>
<tr><td>Operasi</td><td>Isi dari reg₈ posisinya digeser ke kanan 1 bit. Isi bit 0 disalin ke carry flag dan isi bit 7 sebelumnya tidak berubah.</td></tr>
<tr><td>Op Code</td><td>11001011 : 00101[reg₈]
<table>
<tr><th>Register</th><th>Bit</th></tr>
<tr><td>A</td><td>111</td></tr>
<tr><td>B</td><td>000</td></tr>
<tr><td>C</td><td>001</td></tr>
<tr><td>D</td><td>010</td></tr>
<tr><td>E</td><td>011</td></tr>
<tr><td>H</td><td>100</td></tr>
<tr><td>L</td><td>101</td></tr>
</table></td></tr>
</table>

	(HL) | 110
Flag	S Z berubah sesuai yang diinginkan (ditentukan) H N di-reset P/V merupakan paritas Lihat instruksi untuk C
T State	8 atau 15 (HL)

SRA (reg_{index} + ofs_8)

Operasi	Isi memori pada lokasi ditunjuk oleh reg_{index} ditambah ofs_8 posisinya digeser 1 bit ke kiri. Isi bit 7 disalin ke carry flag dan isi zero ditelatak pada bit 0.
Op Code	[reg_{index}] : 11001011 : [ofs_8] : 00101110 **Register** | **Bit** IX | 11011101 IY | 11111101
Flag	S Z berubah sesuai yang diinginkan (ditentukan) H N di-reset P/V merupakan paritas Lihat instruksi untuk C
T State	23

m. Instruksi SRL

SRL reg_8

<table>
<tr><td>Operasi</td><td>Isi dari reg_8 posisinya digeser ke ke kanan 1 bit. Isi bit 0 disalin ke carry flag dan isi zero diletakan pada bit 7.</td></tr>
<tr><td>Op Code</td><td>11001011 : 00111[reg_8]
<table>
<tr><th>Register</th><th>Bit</th></tr>
<tr><td>A</td><td>111</td></tr>
<tr><td>B</td><td>000</td></tr>
<tr><td>C</td><td>001</td></tr>
<tr><td>D</td><td>010</td></tr>
<tr><td>E</td><td>011</td></tr>
<tr><td>H</td><td>100</td></tr>
<tr><td>L</td><td>101</td></tr>
<tr><td>(HL)</td><td>110</td></tr>
</table></td></tr>
<tr><td>Flag</td><td>S Z berubah sesuai yang diinginkan (ditentukan)
H N di-reset
P/V merupakan paritas
Lihat instruksi untuk C</td></tr>
<tr><td>T State</td><td>8 atau 15 (HL)</td></tr>
</table>

SRL (reg_{index} + ofs_8)

Operasi	Isi memori pada lokasi yang ditunjuk reg_{index} ditambah ofs_8 posisinya digeser 1 bit ke kanan.Isi dari bit 0 disalin ke carry flag dan isi zero diletakan ke bit 7.
Op Code	[reg_{index}] : 11001011 : [ofs_8] : 00111110

<table>
<tr><td></td><td><table><tr><td>Register</td><td>Bit</td></tr><tr><td>IX</td><td>11011101</td></tr><tr><td>IY</td><td>11111101</td></tr></table></td></tr>
<tr><td>Flag</td><td>S Z berubah sesuai yang diinginkan (ditentukan)
H N di-reset
P/V merupakan paritas
Lihat instruksi untuk C</td></tr>
<tr><td>T State</td><td>23</td></tr>
</table>

5. Instruksi Kontrol Z80

Instruksi kontrol merupakan instruksi kendali terkait dengan layanan hardware terhadap suatu program, sehingga dengan instruksi ini programer dapat menentukan layanan apa yang diinginkannya.

Adapun instruksi kontrol ini meliputi **CALL, DJNZ, JP, JR, NOP, RET, RST.**

a. Instruksi CALL

CALL imm_{16}

Operasi	Isi PC ditambah tiga diletakan pada stack, kemudian diisi dengan imm_{16}.
Op Code	11001101 : imm_{LSB} : imm_{MSB}
T State	17

CALL cc,imm_{16}

<table>
<tr><td>Operasi</td><td>Jika kondisi cc adalah true, maka isi PC ditambah tiga serta diletakan pada stack, kemudian diisi dengan imm_{16}.</td></tr>
<tr><td>Op Code</td><td>11[cc]100 : imm_{LSB} : imm_{MSB}
<table>
<tr><th>Kondisi</th><th>Bit</th></tr>
<tr><td>NZ</td><td>000</td></tr>
<tr><td>Z</td><td>001</td></tr>
<tr><td>NC</td><td>010</td></tr>
<tr><td>C</td><td>011</td></tr>
<tr><td>PO</td><td>100</td></tr>
<tr><td>PE</td><td>101</td></tr>
<tr><td>P</td><td>110</td></tr>
<tr><td>M</td><td>111</td></tr>
</table>
</td></tr>
<tr><td>T State</td><td>Jika cc adalah true: 17
Jika cc adalah false: 10</td></tr>
</table>

b. Instruksi DJNZ

DJNZ imm_8

Operasi	Dekremen pada register B, dan jika tidak nol, maka nilai imm_8 ditambahkan ke PC. Lompat diukur dari alamat op code instruksi berada.
Op Code	00010000 : [imm_8]
T State	Jika B is not 0: 13

	Jika B is 0: 8

c. Instruksi JP

JP imm_{16}

Operasi	imm_{16} disalin ke PC.
Op Code	11000011 : [imm_{LSB}] : [imm_{MSB}]
T State	10

JP cc,imm_{16}

<table>
<tr><td>Operasi</td><td>Jika kondisi cc adalah true, imm_{16} disalin ke PC.</td></tr>
<tr><td>Op Code</td><td>11[cc]010 : [imm_{LSB}] : [imm_{MSB}]
<table>
<tr><th>Kondisi</th><th>Bit</th></tr>
<tr><td>NZ</td><td>000</td></tr>
<tr><td>Z</td><td>001</td></tr>
<tr><td>NC</td><td>010</td></tr>
<tr><td>C</td><td>011</td></tr>
<tr><td>PO</td><td>100</td></tr>
<tr><td>PE</td><td>101</td></tr>
<tr><td>P</td><td>110</td></tr>
<tr><td>M</td><td>111</td></tr>
</table></td></tr>
<tr><td>T State</td><td>10</td></tr>
</table>

d. Instruksi JR

JR imm$_8$

Operasi	Nilai imm$_8$ ditambahkan ke PC. Lompat diukur dari alamat op code instruksi berada.
Op Code	00011000 : [imm$_8$]
T State	12

JR cc,imm$_8$

<table>
<tr><td>Operasi</td><td>Jika kondisi cc adalah true, maka nilai imm$_8$ ditambahkan ke PC. Lompat diukur dari alamat op code instruksi berada.</td></tr>
<tr><td>Op Code</td><td>001[cc]000
<table>
<tr><th>Kondisi</th><th>Bit</th></tr>
<tr><td>NZ</td><td>00</td></tr>
<tr><td>Z</td><td>01</td></tr>
<tr><td>NC</td><td>10</td></tr>
<tr><td>C</td><td>11</td></tr>
</table></td></tr>
<tr><td>T State</td><td>Jika cc adalah true: 12
Jika cc adalah false: 7</td></tr>
</table>

e. Instruksi NOP

NOP

Operasi	Tidak ada operasi.
Op Code	00000000
T State	4

f. Instruksi RET

RET

Operasi	Isi stack tertinggi diletakan ke PC.
Op Code	11001001
T State	10

RET cc

<table>
<tr><td>Operasi</td><td>Jika kondisi cc adalah true, Isi stack tertinggi diletakan ke PC.</td></tr>
<tr><td>Op Code</td><td>11[cc]000
<table>
<tr><th>Kondisi</th><th>Bit</th></tr>
<tr><td>NZ</td><td>000</td></tr>
<tr><td>Z</td><td>001</td></tr>
<tr><td>NC</td><td>010</td></tr>
<tr><td>C</td><td>011</td></tr>
<tr><td>PO</td><td>100</td></tr>
<tr><td>PE</td><td>101</td></tr>
<tr><td>P</td><td>110</td></tr>
</table>
</td></tr>
</table>

	M | 111
T State	Jika cc adalah true: 11 Jika cc adalah false: 5

g. Instruksi RET

RETI

Operasi	Akhir rutin layanan maskable interrupt (MI) digunakan, isi stack tertinggi diletakan ke PC, dan sinyal pada devais I/O yang telah menyelesaikan interupsi, nested interupsi dimungkinkan (tidak terkait dengan TI).
Op Code	11101101 : 01001101
T State	14

RETN

Operasi	Akhir rutin layanan non-maskable interrupt (alamat pada $0066) digunakan untuk meletakan isi stack tertinggi ke PC. Nilai dari IFF2 disalin ke FF1 sehingga maskable interrupt dimungkinkan dilanjutkan sebagaimana sebelumnya. NMI kondisinya not enable pada TI.
Op Code	11101101 : 01000101
T State	14

h. Instruksi RST

RST imm_8

<table>
<tr><td>Operasi</td><td>Nilai isi PC ditambah tiga dan dituliskan pada stack. MSB diisi dengan $00 dan LSB diisi dengan imm_8.</td></tr>
<tr><td>Op Code</td><td>11[imm_8]111
<table>
<tr><th>Address</th><th>Bit</th></tr>
<tr><td>00h</td><td>000</td></tr>
<tr><td>08h</td><td>001</td></tr>
<tr><td>10h</td><td>010</td></tr>
<tr><td>18h</td><td>011</td></tr>
<tr><td>20h</td><td>100</td></tr>
<tr><td>28h</td><td>101</td></tr>
<tr><td>30h</td><td>110</td></tr>
<tr><td>38h</td><td>111</td></tr>
</table></td></tr>
<tr><td>T State</td><td>11</td></tr>
</table>

6. Instruksi Hardware Z80

Instruksi ini merupakan instruksi Z80 yang secara khusus dirancang untuk mengendalikan atau mengakses perangkat keras yang di dalam mikroprosesor, dengan demikian keinginan programer untuk bisa memanfaatkan komponen mikroprosesor dapat terpenuhi.

Sedangkan kelompok instruksi ini terkait dengan komponen input/output, sistem interupsi dan menghentikan pelaksanaan proses dari mikroprosesor, adapun instruksinya meliputi: **DI, EI, HALT, IM, IN, IND, INDR, INI, INIR, OTDR, OTIR, OUT, OUTD, OUTI.**

- **Instruksi DI**

DI

Operasi	Reset kedua flip-flop interupsi, yaitu menjaga interupsi maskable dari triger.
Op Code	11110011
T State	4

b. Instruksi EI

EI

Operasi	Set kedua flip-flop interupsi, yaitu mengijinkan interupsi maskable untuk tampil. Sebuah interupsi tidak akan tampil sampai selesainya pelaksanaan sebuah instruksi.
Op Code	11111011
T State	4

c. Instruksi HALT

HALT

Operasi	Menghentikan operasi CPU sampai sebuah interupsi atau adanya reset.
Op Code	01110110
T State	4

d. Instruksi IM

Merupakan kelompok instruksi yang dapat digunakan untuk akses perangkat keras terkait dengan sistem interupsi pelaksanaan program, yaitu meliputi:

IM 0

Operasi	Set interupsi pada mode 0.
Op Code	11101101 : 01000110
T State	8

IM 1

Operasi	Set interupsi pada mode 1.
Op Code	11101101 : 01010110
T State	8

IM 2

Operasi	Set interupsi pada mode 2.
Op Code	11101101 : 01011110
T State	4

e. Instruksi IN

Merupakan kelompok instruksi yang dapat digunakan untuk akses perangkat keras terkait dengan sistem baca dan tulis port input/output, yaitu meliputi:

IN A,(imm_8)

Operasi	Satu byte dari port imm_8 diisikan ke A.
Op Code	11011011 : [imm_8]
T State	11

IN reg_8,(C)

<table>
<tr><td>**Operasi**</td><td>Satu byte dari port C diisikan ke reg_8.</td></tr>
<tr><td>**Op Code**</td><td>11101011 : 01[reg_8]000

Register	Bit
A	111
B	000
C	001
D	010
E	011
H	100
L	101

</td></tr>
<tr><td>**Flag**</td><td>S Z berubah sesuai yang diinginkan (ditentukan)
H N di-reset
P/V merupakan paritas
C tidak terpengaruh</td></tr>
<tr><td>**T State**</td><td>12</td></tr>
</table>

f. Instruksi IND

IND

Operasi	Satu byte dari port C diisikan ke lokasi memori locati ditunjuk oleh HL. HL dan B dikurangi 1 (dekremen)
Op Code	11101101 : 10101010
Flag	S H P/V bernilai acak Z di-set jika B menjadi nol (zero)

	N di-set C tidak terpengaruh
T State	16

g. Instruksi INDR

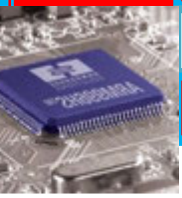
INDR

Operasi	Satu byte dari port C disikan ke lokasi memori yang ditunjuk oleh HL. HL dan B dikurangi 1 (dekremen). Jika B tidak sama dengan nol, maka operasi diulangi.
Op Code	11101101 : 10111010
Flag	S H P/V isinya diacak Z N di-set C tidak terpengaruh
T State	Jika B! = 0: 21 Jika B == 0: 16

i. Instruksi INI

INI

Operasi	Satu byte dari port C disikan ke lokasi memori yang ditunjuk oleh HL dan HL inkremen untuk B dikurangi 1.
Op Code	11101101 : 10100010
Flag	S H P/V isinya diacak Z di-set jika B menjadi nolN di-set C tidak terpengaruh
T State	16



j. Instruksi INIR

INIR

Operasi	Satu byte dari port C disikan ke lokasi memori yang ditunjuk oleh HL. HL inkremen dan B is dekremen. Jika B tidak sama dengan nol, maka operasi diulangi.
Op Code	11101101 : 10110010
Flag	S H P/V isinya diacak Z N di-et C tidak terpengaruh
T State	Jika B != 0: 21 Jika B == 0: 16

k. Instruksi OTDR

OTDR

Operasi	Satu byte dari lokasi memori yang ditunjuk oleh HL disikan ke port C. HL dan B dikurangi 1. Jika B tidak sama dengan nol, maka operasi diulangi.
Op Code	11101101 : 10111011
Flag	S H P/V isinya diacak Z N di-set C tidak terpengaruh
T State	Jika B != 0: 21 Jika B == 0: 16

l. Instruksi OTIR

OTIR

Operasi	Satu byte dari lokasi memori yang ditunjuk oleh HL disikan ke port C. HL inkremen dan B dekremen. Jika B tidak sama dengan nol, maka operasi diulangi
Op Code	11101101 : 10110011
Flag	S H P/V isinya diacak Z N di-set C tidak terpengaruh
T State	Jika B != 0: 21 Jika B == 0: 16

m. Instruksi OUT

Instruksi OUT merupakan instruksi untuk mengeluarkan data dari mikroprosesor kelauar melalui saluran port input/output, ragam dari instruksi ini meliputi deskripsi berikut:

OUT (imm$_8$),A

Operasi	Nilai dari A dituliskan ke port imm$_8$.
Op Code	11010011 : [imm$_8$]
T State	11

OUT (C),reg$_8$

<table>
<tr><td>Operasi</td><td>Nilai dari reg$_8$ dituliskan ke port C.</td></tr>
<tr><td>Op Code</td><td>11101011 : 01[reg$_8$]001
<table><tr><th>Register</th><th>Bit</th></tr><tr><td>A</td><td>111</td></tr><tr><td>B</td><td>000</td></tr><tr><td>C</td><td>001</td></tr><tr><td>D</td><td>010</td></tr><tr><td>E</td><td>011</td></tr><tr><td>H</td><td>100</td></tr><tr><td>L</td><td>101</td></tr></table></td></tr>
<tr><td>T State</td><td>12</td></tr>
</table>

n. Instruksi OUTD

OUTD

Operasi	Satu byte dari lokasi memori yang ditunjuk oleh HL disikan ke port C. HL dan B dikurangi 1.
Op Code	11101101 : 10101011
Flag	S H P/V isi diacak Z di-set jika B menjadi nol N di-set C tidak terpengaruh
T State	16

OUTI

Operasi	Satu byte dari lokasi memori yang ditunjuk oleh HL disikan ke port C. HL inkremen dan B dekremen.
Op Code	11101101 : 10100011
Flag	S H P/V isi diacak Z di-set jika B menjadi nol N di-set C tidak terpengaruh
T State	16

7. Instruksi Set

Merupakan kelompok instruksi yang dapat digunakan untuk men-set isi register, instruksi ini lebih ditekankan pada alternatip lebih baik dari pelaksanaan instruksi yang telah dijelaskan sebelumnya. Kelompok instruksi ini meliputi: **IXH**, **IXL**, **IYH**, **IYL**, **IN**, **OUT**, **SLL**, **Autocopy**.

a. Instruksi IXH Register

Pelaksanaan instruksi dengan DB $DD merupakan operasi instruksi mengguna-kan MSB dari register IX, hal ini lebih baik dibanding harus menggunakan register H.

- **ADD A,H**
- **AND H**
- **CP H**
- **DEC H**
- **INC H**
- **LD reg_8,H**
- **LD H,reg_8**
- **LD H,imm_8**
- **OR H**
- **SBC A,H**

- **SUB H**
- **XOR H**

- **Instruksi IXL Register**

Pelaksanaan instruksi dengan.DB $DD merupakan operasi instruksi menggunakan LSB register IX, hal ini lebih baik dibanding harus menggunakan register L yaitu meliputi instruksi:

- **ADD A,L**
- **AND L**
- **CP L**
- **DEC L**
- **INC L**
- **LD reg_8,L**
- **LD L,reg_8**
- **LD L,imm_8**
- **OR L**
- **SBC A,L**
- **SUB L**
- **XOR L**

c. Instruksi IYH Register

Pelaksanaan instruksi dengan.DB $FD merupakan operasi instruksi menggunakan MSB register IY, hal ini lebih baik dibanding harus menggunakan register H

- **ADD A,H**
- **AND H**
- **CP H**
- **DEC H**
- **INC H**
- **LD reg_8,H**

- **LD H,reg$_8$**
- **LD H,imm$_8$**
- **OR H**
- **SBC A,H**
- **SUB H**
- **XOR H**

d. Instruksi IYL Register

Pelaksanaan instruksi dengan DB $FD merupakan operasi instruksi menggunakan LSB register IY, hal ini lebih baik dibanding harus menggunakan register L

- **ADD A,L**
- **AND L**
- **CP L**
- **DEC L**
- **INC L**
- **LD reg$_8$,L**
- **LD L,reg$_8$**
- **LD L,imm$_8$**
- **OR L**
- **SBC A,L**
- **SUB L**
- **XOR L**

e. Instruksi IN

IN (C)

Operasi	Input satu byte dari port C dan berpengaruh hanya pada Flag.
Op Code	11101101 : 01110000
Flag	S Z berubah sesuai yang diinginkan (ditentukan) H N di-reset

	P/V merupakan paritas C tidak terpengaruh
T State	12

f. Instruksi OUT

OUT (C),0

Operasi	Keluarkan nol ke port C.
Op Code	11101101 : 01110001
T State	12

g. Instruksi SLL

SLL reg_8

<table>
<tr><td>Operasi</td><td>Isi dari reg_8 digeser ke kiri bit positip. Isi dari bit 7 diletakan ke dalam carry flag dan isi carry diletakan ke dalam bit 0.</td></tr>
<tr><td>Op Code</td><td>11001011 : 00110[reg_8]
<table>
<tr><th>Register</th><th>Bit</th></tr>
<tr><td>A</td><td>111</td></tr>
<tr><td>B</td><td>000</td></tr>
<tr><td>C</td><td>001</td></tr>
<tr><td>D</td><td>010</td></tr>
<tr><td>E</td><td>011</td></tr>
<tr><td>H</td><td>100</td></tr>
<tr><td>L</td><td>101</td></tr>
</table></td></tr>
</table>

	(HL)	110
Flag	S Z berubah sesuai yang diinginkan (ditentukan) H N di-reset P/V merupakan paritas Lihat instruksi untuk C	
T State	8 atau 15 (HL)	

SLL (reg_{index} + ofs_8)

Operasi	Isi memori pada lokasi ditunjuk oleh reg_{index} ditambah ofs_8 digeser ke kiri pada bit positip. Isi dari bit 7 diletakan ke dalam carry flag dan isi carry diletakan ke dalam bit 0.
Op Code	[reg_{index}] : 11001011 : [ofs_8] : 00110110
Flag	S Z berubah sesuai yang diinginkan (ditentukan) H N di-reset P/V merupakan paritas Lihat instruksi untuk C
T State	23

Register	Bit
IX	11011101
IY	11111101

h. Instruksi Autocopy.

Setiap instruksi autocopy digunakan untuk operasi byte dalam memori melalui penunjukan register indeks. Setelah operasi ini, hasil diletakan pada register 8-bit dan register tetap memegang hasil modifikasi.

RL $reg_8,(reg_{index} + ofs_8)$

RLC $reg_8,(reg_{index} + ofs_8)$

RR $reg_8,(reg_{index} + ofs_8)$

RRC $reg_8,(reg_{index} + ofs_8)$

SLA $reg_8,(reg_{index} + ofs_8)$

SLL $reg_8,(reg_{index} + ofs_8)$

SRA $reg_8,(reg_{index} + ofs_8)$

SRL $reg_8,(reg_{index} + ofs_8)$

RES $reg_8,imm_3,(reg_{index} + ofs_8)$

SET $reg_8,imm_3,(reg_{index} + ofs_8)$

Opcode dari instruksi autocopy dibentuk melalui cara berikut:

1. Ambil op-code untuk versi non-autocopy, contoh permasalahan dari **RL C,(IX + 2)**, op-codenya adalah untuk **RL (IX + 2)**: 11011101 : 11001011 : 00000010 : 00010110
2. Ubah 3 bit terakhir dari byte terakhir op-code dengan Bit untuk reg_8 berikut:

Register	**Bit**
A	111
B	000
C	001
D	010
E	011

H	100
L	101
(reg_{index}) (no autocopy)	110

Terakhir op-code untuk **RL C,(IX + 2)** adalah:

11011101 : 11001011 : 00000010 : 00010001

4.3. OPERASI INSTRUKSI MIKROPROSESOR Z80.

1. SISTEM / CARA PENGALAMATAN

Kejelasan yang sistimatik tentang cara pengalamatan sangat penting pada pengolahan data dalam jenis Prosesor, sebab program yang disusun tanpa penggunaan pengalamatan yang pasti, maka program tersebut menjadi kurang efektif dalam penganalisaannya.

Semakin panjang kode operasi sebuah perintah, maka dapat juga dikombinasikan banyak cara pengalamatannya.

Pada dasarnya cara pengalamatannya dapat dibagi menjadi 4 cara yang berbeda, yaitu meliputi:

- Immediate (Segera), kode mesin mengandung konstanta untuk segera /langsung di akses
- Direct (Langsung), kode mesin mengandung Register, alamat penyimpan atau alamat masukan / keluaran dari operan untuk diakses
- Indirect (Tidak Langsung), kode mesin mengandung hanya satu petunjuk, dimana untuk mendapatkan alamat dari operan yang akan diakses
- Terindeks, alamat-alamat dari operan yang akan di akses dibentuk dalam beberapa bagian.

a. Pengalamatan Immediate

Disini operan yang akan diakses langsung terkandung pada kode mesin, ini adalah cara yang sangat sederhana, untuk mengisi konstanta ke Register atau lokasi penyimpanan . Tentu saja operan tidak dapat diubah lagi, maka kode mesin yang demikian kebanyakan disimpan di ROM . Kode operasi hanya dapat mengandung satu petunjuk tentang panjang dari operan yang mengikutinya . Selain itu bagian alamat masih harus mengandung sebuah keterangan tentang tujuan dimana konstanta harus dihubungkan kepadanya.

Contoh :

LD HL, 1234H

adr	0	0	1	0	0	0	0	1	tujuan adalah Register HL
adr + 1	0	0	1	1	0	1	0	0	bagian kanan dari Konstanta
adr + 2	0	0	0	1	0	0	1	0	bagian kiri dari Konstanta

ADD A, 12H

adr	1	1	0	0	0	1	1	0	pada operasi arithmatik dengan operasi 8 bit, tujuan salalu A
adr + 1	0	0	0	1	0	0	1	0	Konstanta

LD (HL), 12H

adr	0	0	1	1	0	1	1	0	Tujuan adalah lokasi penyimpanan yg dialamatkan melalui HL
adr + 1	0	0	0	1	0	0	1	0	Konstanta

JP 0916H

adr	1	1	0	0	0	0	1	1	Konstanta 16 BIT yang mengikuti op-kode (mempunyai sifat
adr + 1	0	0	0	1	0	1	1	0	sebagai operan yang dapat segera diakses di isi ke PC
adr + 2	0	0	0	0	1	0	0	1	dan dipakai sebagai alamat)

b. Pengalamatan Langsung (Direct)

Disini kode mesin mengandung sebuah atau lebih alamat-alamat yang kemudian isi dari alamat-alamat ini akan diakses lebih lanjut. Panjang alamat-alamat ini dapat berbeda menurut keadaan apakah itu mengenai sebuah Register, alamat penyimpan atau alamat masukan/keluaran, perintah dapat mengandung sebuah petunjuk, apakah bagian pertama diberikan sebagai alamat tujuan atau sumber.

Contoh :

LD A, (1234H)

adr	0	0	1	1	1	0	1	0	
adr + 1	0	0	1	1	0	1	0	0	Alamat sumber penyimpan 16 BIT
adr + 2	0	0	0	1	0	0	1	0	

INC L

adr	0	0	1	0	1	1	0	0	op-kode yang mengandung alamat Register

LD E, C

adr	0	1	0	1	1	0	0	1	dalam perintah ini mengandung 2 alamat Register

OUT (20H), A

adr	1	1	0	1	0	0	1	1	alamat port masukan/keluaran
adr + 1	0	0	1	0	0	0	0	0	

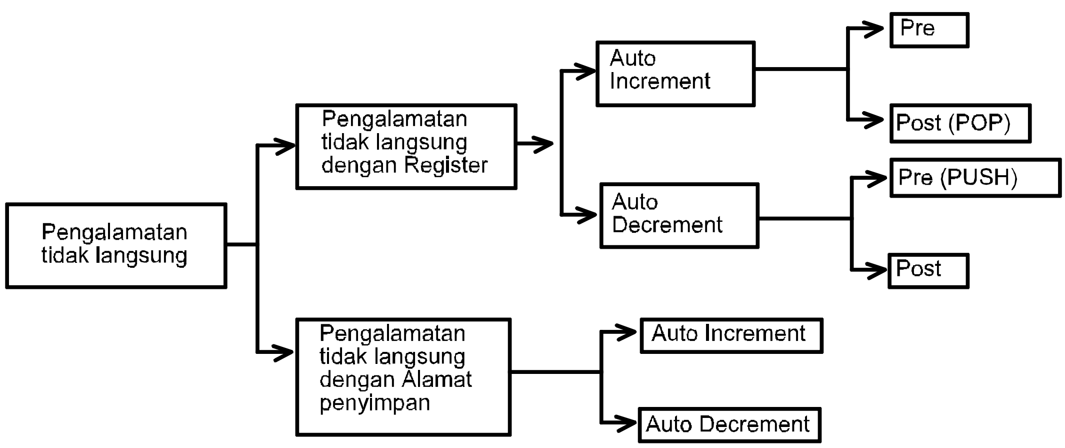
JP (IX)

adr	1	1	0	1	1	1	0	1	loncat ke alamat yang di berikan dalam register IX
adr + 1	1	1	1	0	1	0	0	1	

c. Indirect (Tidak Langsung)

Pengalamatan tidak langsung dapat dilakukan melalui sebuah register atau dapat juga dilakukan dengan penujuk alamat memori, kode mesin hanya mengandung sebuah petunjuk alamat, dimana alamat merupakan operan yang akan diakses. Petunjuk alamat dapat dilakukan melalui sebuah register CPU atau pada sebuah lokasi penyimpan, sehingga secara efektif akan didapatkan pengalamatan secara tidak langsung dengan register dan kode mesin menjadi lebih pendek .

Sebagai contoh sebuah penunjuk alamat dengan alamat 16 BIT dapat diganikan hanya 3 BIT alamat Register. Perintah dengan penunjuk alamat tidak langsung pada Z 80 dapat di bagi seperti gambar berikut :



Gambar 4.3. Pengalamatan tidak langsung

Contoh :

PUSH dan **POP** menaikkan atau menurunkan Register alamat yang dipakai secara otomatis dan terjadi apakah pada sebelum atau sesudah pelaksanaan perintahnya.

RRC (HL)

adr	1	1	0	0	1	0	1	1	petunjuk pada HL Register Alamat
adr + 1	0	0	0	0	1	1	1	0	

PUSH DE

adr	1	1	0	1	0	1	0	1	
									Pengalamatan langsung dari Register DE, isi dari Register SP dipakai sebagai alamat

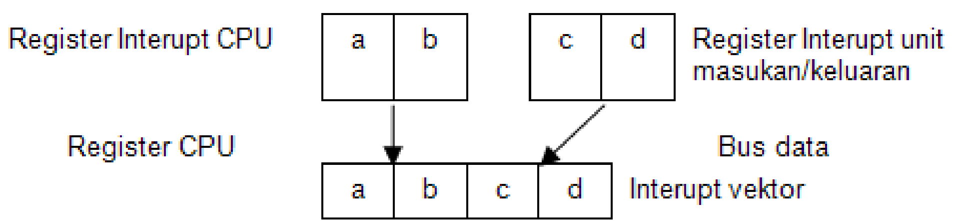
d. Terindeks

- **Pembentukan Alamat Melalui Penyusunan**

Disini alamat efektif disusun dari beberapa bagian yang mana bagian-bagian ini dapat berasal dari Register CPU, Register-Register masukan/keluaran atau dari lokasi penyimpan.

Contoh :

Z 80 menggunakan cara pengalamatan ini pada proses Interrupt dalam mode



RST 5

	7							0
adr	1	1	1	0	1	1	1	1

15															0	
0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	= Alamat loncat 0028H

- **Pembentukan Alamat Melalui Penambahan**

Disini alamat efektif disusun dari beberapa bagian yang mana bagian-bagian ini dapat berasal dari Register-Register CPU, Register masukan/keluaran atau dari lokasi penyimpan.

Contoh :

LD E, (IX + 12)

adr	1	1	0	1	1	1	0	1	Konstanta 12 H ditambah
adr + 1	0	1	0	1	1	1	1	0	dengan isi Register IX
adr + 2	0	0	0	1	0	0	1	0	

JR 12H

adr	0	0	0	1	1	0	0	0	Konstanta 12 H ditambah
adr + 1	0	0	0	1	0	0	1	0	dengan nilai biner adr + 2 dan hasilnya diisi ke PC, PC = adr + 2 + 12 H

1.2.1 Latihan 1

1. Buatlah op-kode pengalamatan langsung dari !

INC D

Adr									⇒ op kode alamat INC Reg.D

2. Buatlah op-kode pengalamatan langsung dari !

RLC (HL)

Adr									⇒ op kode RLC (HL)
adr + 1									

3. Sebutkan beberapa pengalamatan tidak langsung !

1.2.2 Jawaban 1

1. **INC D**

adr									⇒ op kode alamat INC Reg.D

2. **RLC (HL)**

adr									⇒ op kode RLC (HL)
adr + 1									

3. Sebutkan beberapa pengalamatan tidak langsung !

- Pengalamatan tidak langsung dengan Register
  - AUTO INCREMENT :

a) Pre

b) Post (POP)

  - AUTO DECREMENT

a) Pre (Push)

b) Post

- Pengalamatan tidak langsung dengan alamat penyimpan
  - AUTO INCREMENT
  - AUTO DECREMENT

2. PERINTAH TRANSFER

Terdapat beberapa instruksi atau perintah transfer yang digunakan dalam bahasa assembly, berdasarkan strukturnya instruksi transfer meliputi:

⇒ Transfer dengan pengalamatan Immediate

⇒ Transfer dengan pengalamatan langsung

⇒ Transfer dengan pengalamatan tidak langsung melalui pasangan Register

⇒ Transfer dengan pengalamatan tidak langsung melalui pasangan Register + offset

⇒ Transfer dengan pengalamatan Stack

⇒ Transfer dengan pertukaran data

⇒ Untuk input output dengan pengalamatan langsung

⇒ Untuk input output dengan pengalamatan tidak langsung

Untuk tranport data hanya dapat dilaksanakan kemungkinan-kemungkinan dibawah ini :

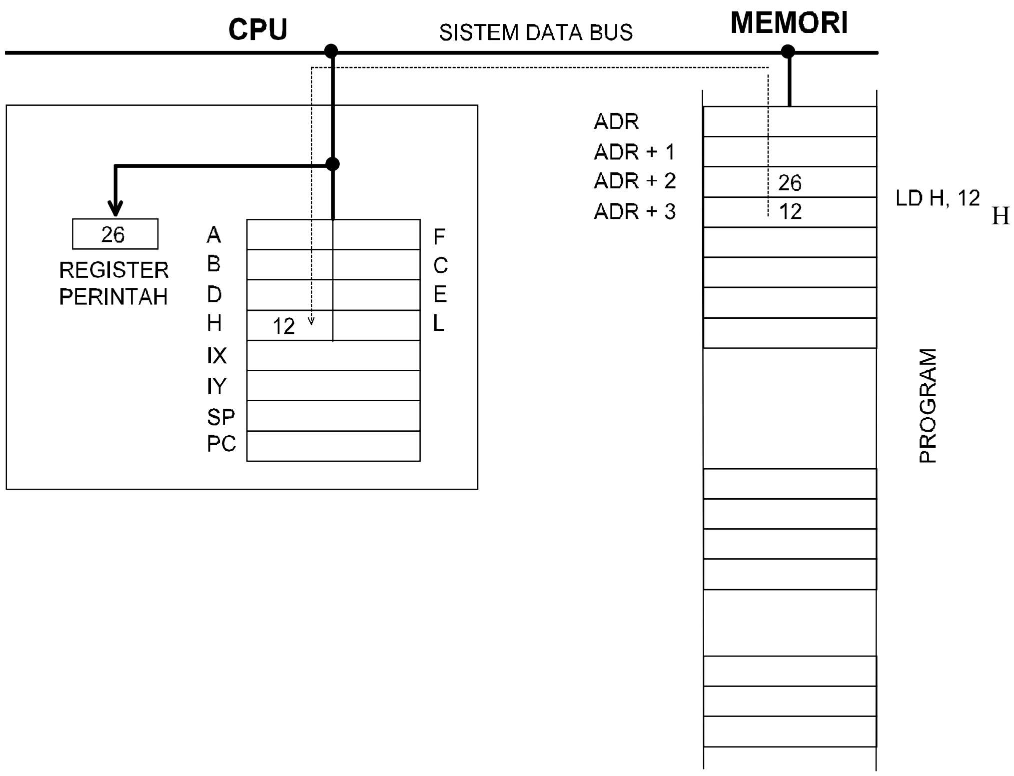
Register CPU ⇔ Register CPU

Register CPU ⇔ Memori

Register CPU ⇔ Register Input – Output

a. Perintah Transfer Dengan Pengalamatan Immediate

Perintah ini melakukan kemungkinan sederhana untuk mengisi Register CPU 8 bit atau 16 bit dengan sebuah konstanta.



Sebuah Register CPU diisi dengan konstanta yang mengikuti op code pada Memori

Mnemonik

LD r,n

r = Register CPU (8 bit) A,B,C,D,E,F,H,L

n = Data (8 bit)

LD rr, nn

rr = Register CPU (16 bit) BC,DE,HL,IX,IY,SP

nn = Data (16 bit)

Operasi

$r \leftarrow n$

$rr \leftarrow nn$

Register CPU r atau rr diisi dengan konstanta n atau nn yang mengikuti kode mesin

Format

LD r, n

adr	0	0		r		1	1	0
adr + 1	**konstanta**							

LD rr, nn

adr	0	0	rr		0	0	0	1
adr + 1	**konstanta LSB**							
adr + 2	**konstanta MSB**							

adr = Alamat Memori

r dan rr= dapat berupa :

A = 111	D = 010	L = 101	BC = 00	SP = 11
B = 000	E = 011		DE = 01	
C = 001	H = 100		HL = 10	

Flag Tidak berpengaruh

b. Perintah Transfer Dengan Pengalamatan Langsung

Pada pengalamatan langsung, alamat sumber dan tujuan, berada pada kode mesin.

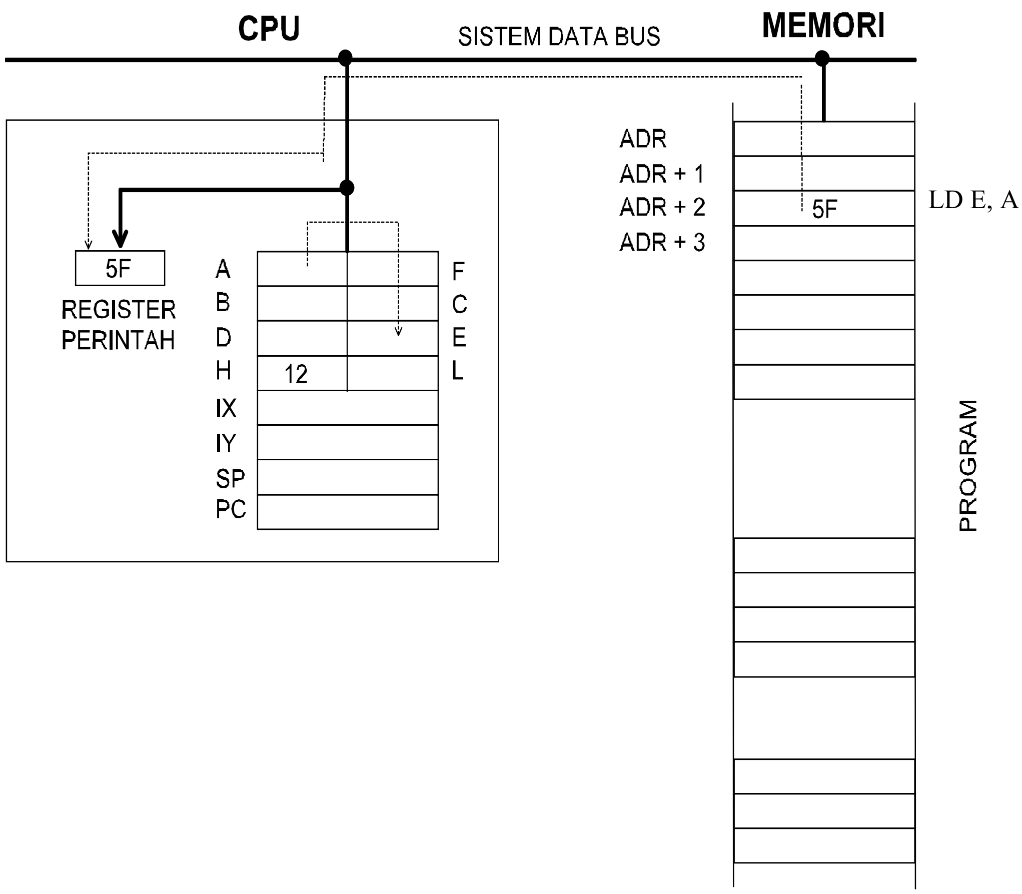
Alamat register atau memori dapat ditulis langsung.

Perintah transfer pengalamatan langsung dapat dibagi menjadi :

- Pengalamatan Register - Register
- Pengalamatan Register – Memori

➢ **Pengalamatan Register - Register**

Perintah ini mengakibatkan transfer data dari sebuah Register CPU ke Register CPU yang lain.



Sebuah Register diisi dengan isi dari Register CPU yang lain.

Mnemonik

LD r,r’

Operasi

r ← r’

Register CPU tujuan r diisi dengan isi Register CPU sumber r’

Format

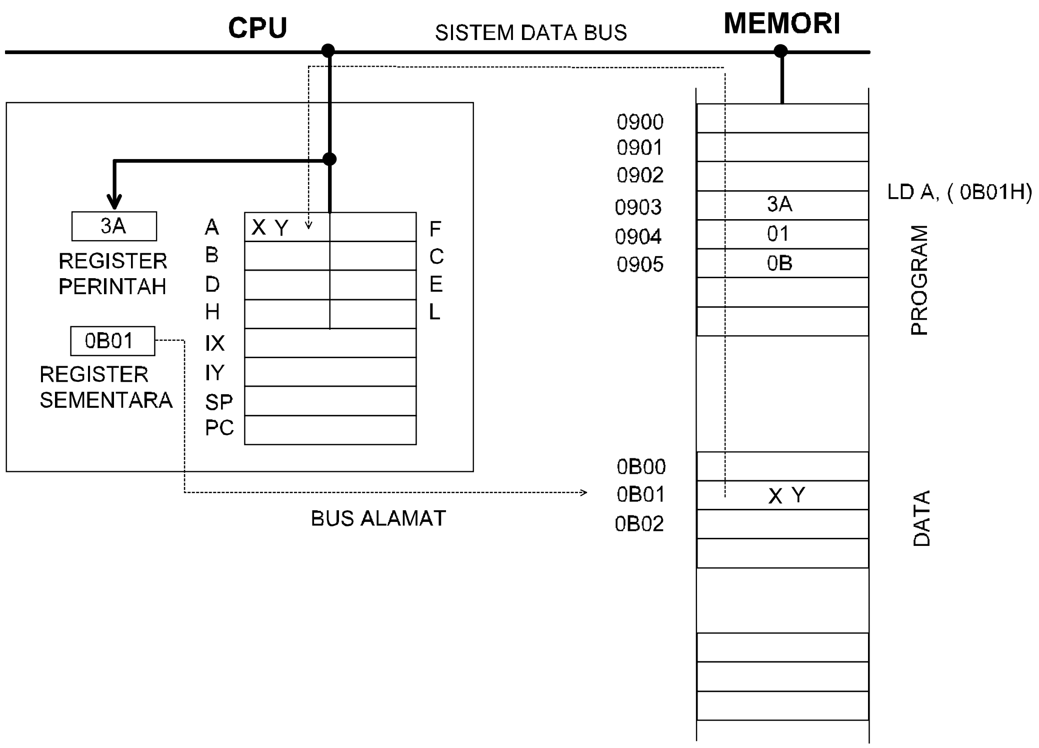
0	1		r			r’	

Flag

Tidak terpengaruh

➢ **Pengalamatan Register - Memori**

Jenis pengalamatan ini, melaksanakan transfer antara Register CPU dan lokasi Memori yang dituliskan di belakang kode mesin.



Sebuah Register CPU diisi dengan isi dari lokasi memori yang dihasilkan di belakang mesin.

Mnemonik

Register CPU ← Memori	Memori ← Register CPU
LD A, (nn)	**LD (nn), A**
LD rr, (nn)	**LD (nn), rr**

rr = Register CPU 16 bit (BC, DE, HL, SP)

(nn) = Isi dari tanda ini selalu diisi oleh alamat Memori dan tanda ini berarti isi dari alamat memori yang ditunjuk oleh nn.

Operasi

Register A CPU ← Memori

Register CPU diisi oleh isi dari lokasi Memori yang alamatnya di tunjuk oleh alamat dalam tanda kurung.

Memori ← Register A CPU

Memori yang alamatnya ditunjuk oleh alamat dalam tanda kurung diisi oleh isi Register A CPU.

Register CPU ← Memori

Register CPU 16 bit diisi oleh isi dari memori yang alamatnya ditunjuk oleh alamat dalam tanda kurung.

Memori ← Register CPU 16 Bit

Lokasi Memori yang alamatnya ditunjuk oleh alamat dalam tanda kurung diisi oleh isi dari Register CPU 16 Bit.

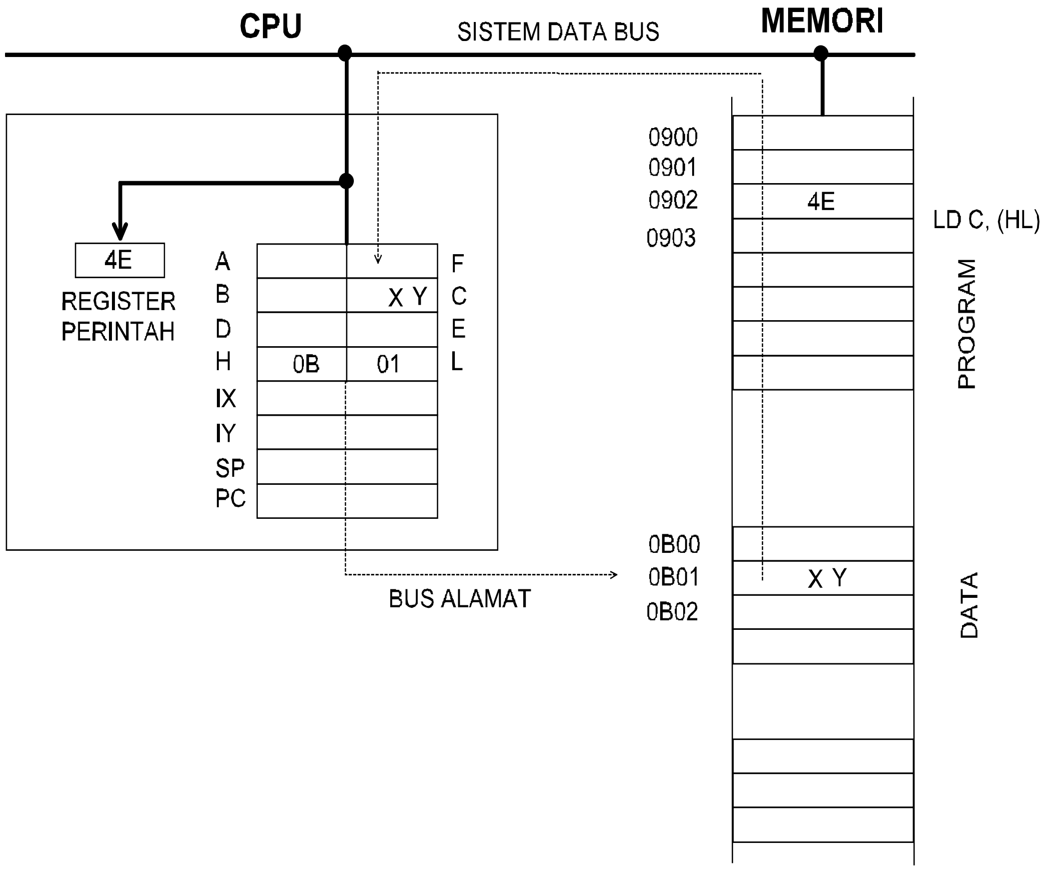
Format

	LD A, (nn)									**LD (nn), A**							
adr	0	0	1	1	1	0	1	0		0	0	1	1	0	0	1	0
adr + 1	**Alamat LSB**									**Alamat LSB**							
adr + 2	**Alamat MSB**									**Alamat MSB**							

	LD rr, (nn)									LD (nn), rr							
adr	1	1	1	0	1	1	0	1		1	1	1	0	1	1	0	1
adr + 1	0	1	rr		1	0	1	1		0	1	Rr		0	0	1	1
adr + 2	**Konstanta LSB**									**Konstanta LSB**							
adr + 3	**Konstanta MSB**									**Konstanta MSB**							

Flag Tidak terpengaruh.

- **Perintah Transfer Pengalamatan Tidak Langsung Melalui Pasangan Register**



Sebuah Register CPU diisi oleh isi sebuah lokasi Memori yang ditunjuk oleh pasangan Register CPU.

Keuntungan dari jenis pengalamatan ini adalah :

- Kode mesin lebih pendek seperti pada pengalamatan langsung sehingga kebutuhan memori program sedikit dan pelaksanaan perintah lebih cepat.
- Data alamat dapat mudah dimanipulasi.

Mnemonik

LD r, (rr) Register CPU $\leftarrow$ Memori

LD (rr), r Memori $\leftarrow$ Register CPU

r = Register CPU 8 bit (A,B,C,D,E,H,L)

rr = Regiser CPU 16 bit

HL : dapat dipergunakan untuk semua Register CPU

BC,DE : Hanya dapat dipergunakan untuk akkumulator

Operasi

r $\leftarrow$ (rr)

Register CPU r diisi oleh isi dari lokasi Memori yang ditunjuk oleh pasangan Register CPU rr

(rr) $\leftarrow$ r

Lokasi Memori yang ditunjuk oleh pasangan Register CPU rr diisi oleh isi Register CPU r

Format

LD (HL), r							
0	1	1	1	0		r	

LD r, (HL)							
0	1		r		1	1	0

LD (BC),A							
0	0	0	0	0	0	1	0

LD A, (BC)							
0	0	0	0	1	0	1	0

LD (DE), A							
0	0	0	1	0	0	1	0

LD A, (DE)							
0	0	0	1	1	0	1	0

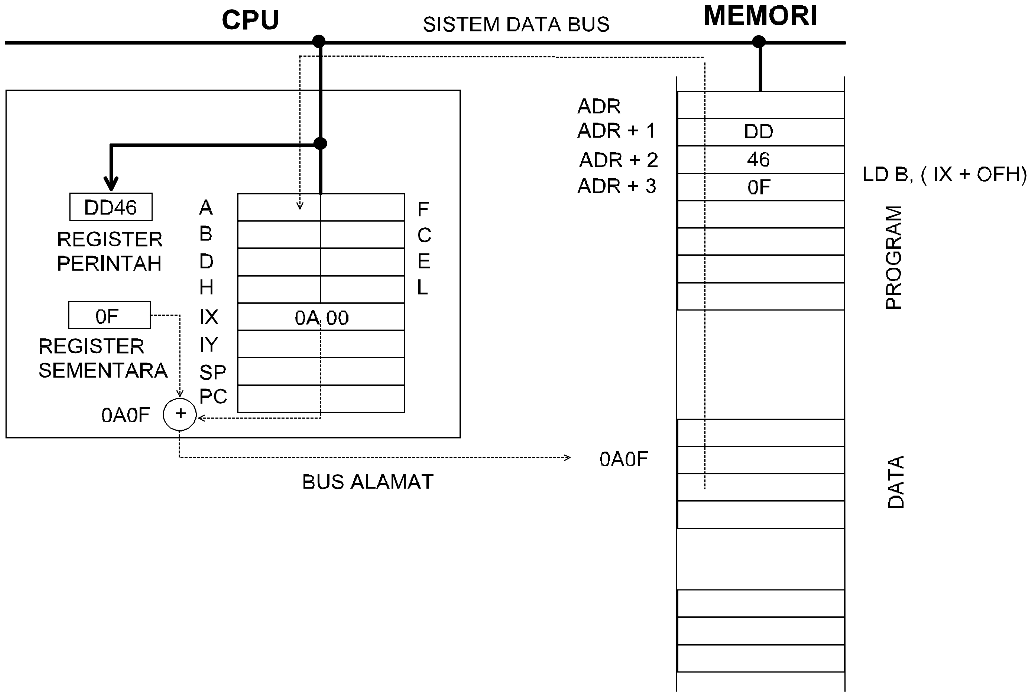
LD (HL), n							
0	0	1	1	0	1	1	0
KONSTANTA							

Flag Tidak terpengaruh.

- **Perintah Transfer Pengalamatan Tidak Langsung Melalui Register + Offset**

Dipergunakan untuk transfer data 8 bit antara Register, CPU dan Memori. Penunjukkan yang tepat sebenarnya adalah Register tidak langsung + offset. Sebagai Register alamatnya biasa dipakai Register index IX dan IY.

Untuk itu Register index ini harus diisi terlebih dahulu dengan alamat basis yang diinginkan.



Mnemonik :

Register CPU ← Memori

LD r, (IR + e)

Memori ← Register CPU

LD (IR + e), r

LD (IR + e), n

r = Register CPU 8 bit (A,B,C,D,E,HL)

IR = Register Index IX dan IY

e = Jarak,offset,konstanta 8 bit

n = Konstanta 8 bit

Operasi

Register CPU ← Memori

Register CPU r diisi oleh isi dari lokasi Memori yang ditunjuk oleh isi Register Index + offset

Memori ← Register CPU

Lokasi memori yang ditunjuk oleh Register Index + offset diisi oleh register CPU r

Format

	LD r, (IX + e)									**LD (IX + e), r**							
adr	1	1	0	1	1	1	0	1		1	1	0	1	1	1	0	1
adr + 1	0	1		r		1	1	0		0	1	1	1	0		r	
adr + 2	konstanta e									konstanta e							

	LD r, (IY + e)									**LD (IY + e), r**							
adr	1	1	1	1	1	1	0	1		1	1	1	1	1	1	0	1
adr + 1	0	1		r		1	1	0		0	1	1	1	0		r	
adr + 2	konstanta e									konstanta e							

LD (IX + e), n							
1	1	0	1	1	1	0	1
0	0	1	1	0	1	1	0
konstanta e							

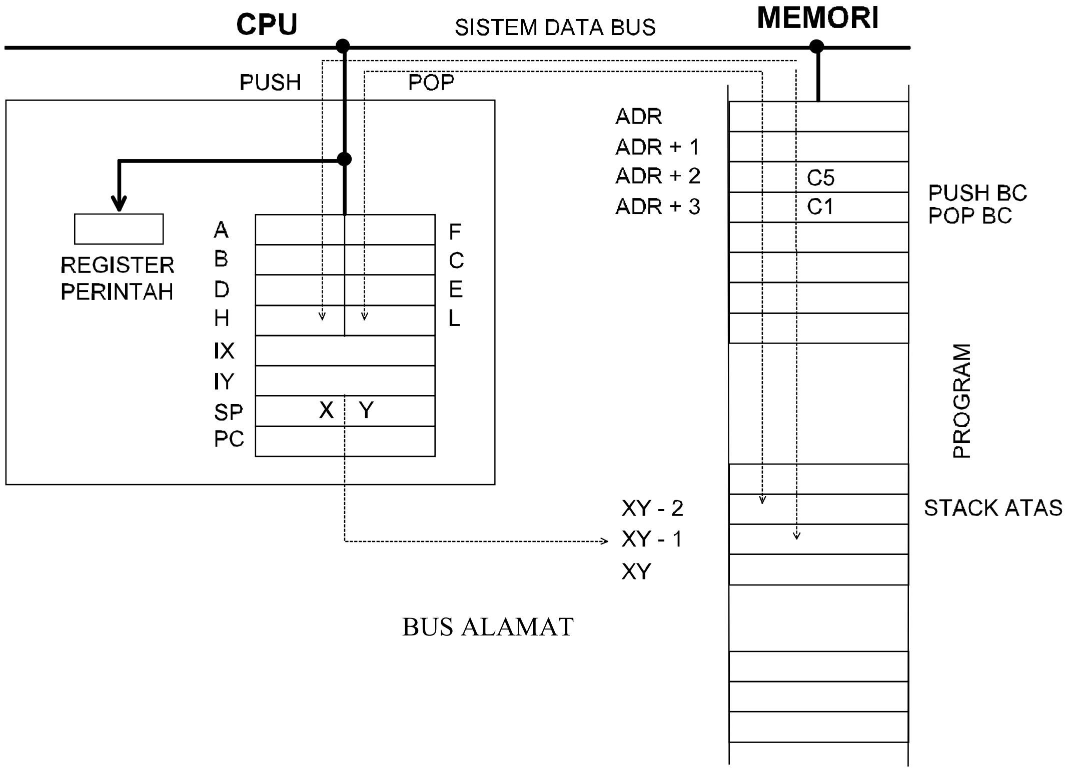
LD (IY + e)							
1	1	1	1	1	1	0	1
0	0	1	1	0	1	1	0
konstanta e							
konstanta n							

- **Perintah Transfer Dengan Pengalamatan Stack**

Perintah ini memungkinkan pemrograman untuk menyimpan isi Register CPU pada Memori sementara dan untuk pemrosesan secara sederhana pada blok data.

Pengalamatan Stack adalah prinsip pada pengalamatan Register CPU 16 bit.

Contoh : Stack pointer (SP, Stack pointer). SP secara otomatis akan dinaikkan atau akan diturunkan 2, setelah pembacaan atau penulisan pada Stack.



Mnemonik

Register CPU ← Memori

POP rr

Memori ← Register CPU

PUSH rr

rr = Register CPU 16 bit (AF,BC,DE,HL,IX,IY)

Operasi :

Register alamat SP ini mempunyai sifat yang sangat praktis yaitu sebelum penyimpanan sebuah byte oleh perintah push, isi dikurangi 1 dan setelah pembacaan sebuah byte oleh perintah POP, isi SP ditambah 1.

Proses penambahan dan pengurangan isi SP dilakukan secara otomatis oleh block pemroses perintah.

Format :

POP rr								**PUSH rr**						
1	1	rr	0	1	0	1		1	1	rr	0	0	0	1

PUSH	**POP**
SP: = SP -1	Register 16 bit rendah : = (SP)
(SP) = Register 16 bit tinggi	SP : = SP + 1
SP : = SP -1	Register 16 bit tinggi : = (SP)
(SP) Register 16 bit lebih rendah	SP : = SP + 1

SP selalu berisikan alamat Memori terakhir yang sedang aktif setelah pelaksanaan perintah PUSH atau POP.

Contoh :

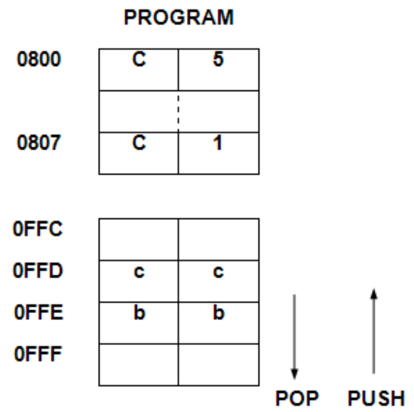
Register CPU BC berisikan konstanta bbccH

PUSH BC isi BC di simpan pada Stack

POP BC isi Stack di ambil dari stack

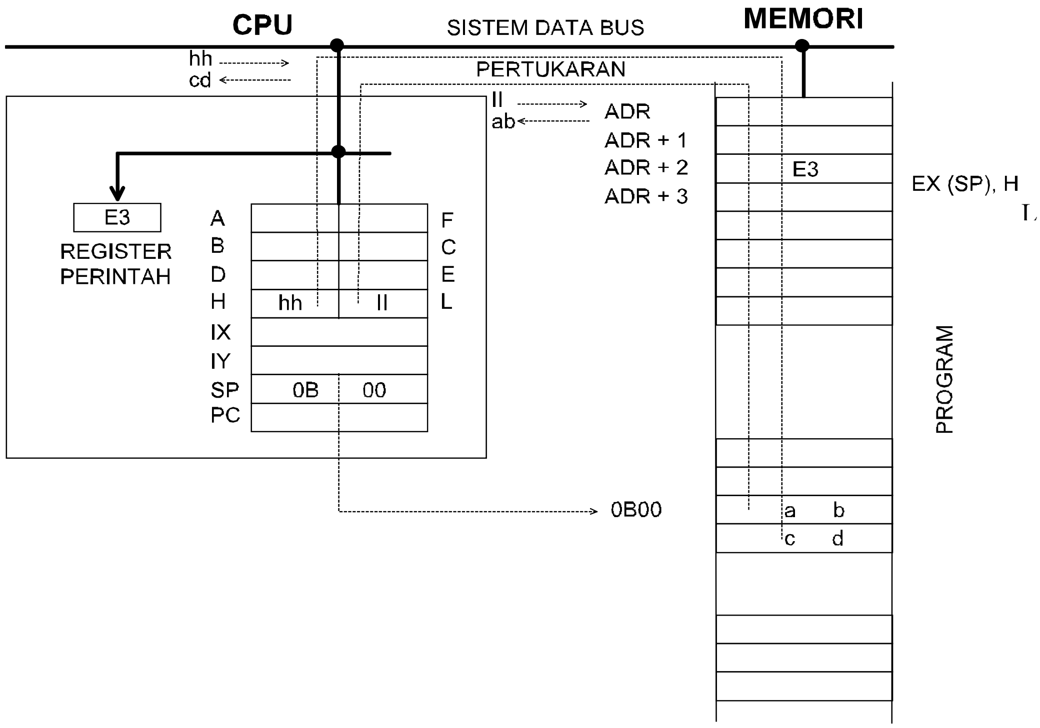
SP setelah pelaksanaan PUSH

SP sebelum pelaksanaan POP



➢ Perintah Transfer Dengan Pertukaran Data

Dengan kelompok perintah ini, tidak sama seperti register tujuan di isi dengan Register sumber, (isi pada register sumber tidak berubah), tetapi pada perintah ini, isi kedua Register saling bertukar.



Mnemonik :

EX DE, HL

EX (SP), HL

Format :

EX DE, HL									EX (SP), HL							
1	1	1	0	1	0	1	1		1	1	1	0	0	0	1	1

Operasi :

Isi pasangan Register saling dipertukarkan

Flag

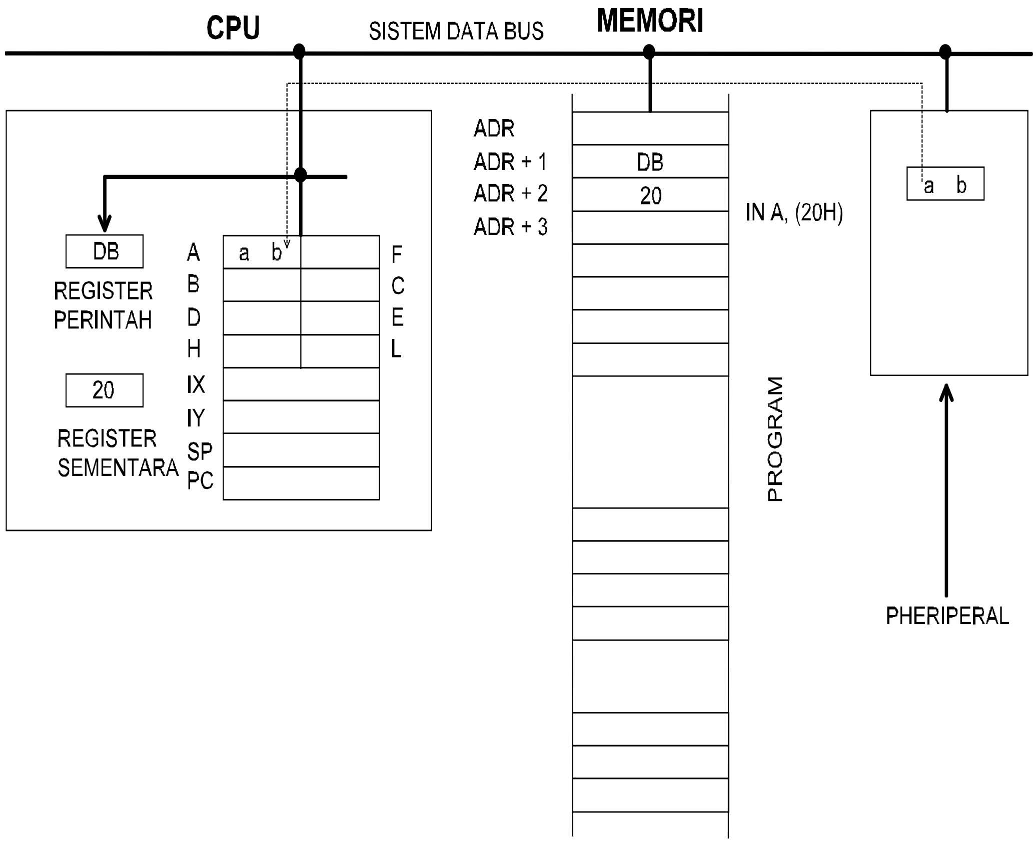
Tidak terpengaruh

c. Perintah Untuk Input Output Dengan Pengaturan Langsung

Perintah ini mengontrol lalu lintas bus data antara CPU dan piranti input output. Pada sebagian sistim mikroprosessor sering dilengkapi dengan banyak blok input output dengan sebuah atau lebih register, yang melalui fungsi blok input output ini dipakai sebagai pelayanan penyangga data pada Register sementara atau pengaturan informasi kontrol.

Register pada piranti ini,yang dipakai sebagi penghubung sistim dengan dunia luar (peripheral) disebut sebagai “port”, dan alamatnya disebut “alamat port”. Jumlah alamat yang dipakai oleh perintah input output ini hanya 8 bit dan dalam pelaksanaannya diberikan melalui jalur penghantar A7 - A0.

Dengan demikian dapat dibentuk 256 alamat port yang berbeda.



MNEMONIK :

CPU ← Register I/O

IN A, (n)

Register I/O ← CPU

OUT (n), A

n : konstanta 8 bit

Format :

	IN A, (n)									**OUT (n), A**							
adr	1	1	0	1	1	0	1	1		1	1	0	1	0	0	1	1
adr + 1	**Konstanta**									**Konstanta**							

Operasi :

A, (n)

Akku CPU diisi dengan isi register I/O yang beralamat n

(n), A

Register I/O yang beralamat n diisi oleh isi dari Akku CPU

Flag

Tidak terpengaruh

CONTOH :

IN A,(20H) 0900 DB

0901 20

d. Perintah Input Output Dengan Pengalamatan Tidak Langsung

Mnemonik :

CPU Register I/O

IN r, (C)

Piranti I/O CPU

OUT (C), r

r = Register 8 bit A,B,C,D,E,H,L



C = Register C yang isinya diberikan sebagai penunjuk penghantar alamat A7 - A0

Format

	IN r, (C)									**OUT (C), r**							
adr	1	1	1	0	1	1	0	1		1	1	1	0	1	1	0	1
adr + 1	0	1		r		0	0	0		0	1		r		0	0	1

Operasi

Register CPU tujuan r diisi dengan isi dari Register C yang merupakan pengalamatan dari port I/O

Register C yang merupakan pengalamatan dari port I/O diisi dengan isi dari Register CPU r.

Flag :

Pada perintah input (IN)

Flag S = 1, bila bit tertinggi = 1

Flag Z = 1, bila data yang dibaca = 0

Flag P = 1, pada parity genap dari data yang di baca

1.2.3 Latihan 2

1. Mengisi Register B dengan D8H

   a). Mnemonik :

   b). Bahasa Mesin :

2. Memindahkan isi Register B ke Register H, bila isi Register B = D8H

   a). Mnemonik :

   b). Bahasa mesin :

3. Memindahkan isi register A ke lokasi Memori beralamat 0900H

   a). Mnemonik :

   b). Bahasa mesin :

4. Memindahkan isi lokasi Memori ber alamat 0904H ke Register A

   a). Mnemonik :

   b). Bahasa mesin :

5. Memindahkan isi Register B ke lokasi Memori yang alamatnya ditunjuk oleh Register HL, bila isi Register HL = 0908H

   a). Mnemonik :

   b). Bahasa mesin :

6. Memindahkan isi lokasi Memori yang alamatnya ditunjuk oleh Register HL ke register C, bila isi lokasi Memori beralamat 0907H = 09H

   a). Mnemonik :

   b). Bahasa mesin :

7. Memindahkan isi Register A ke Register Port I/O yang alamatnya 00H

   a). Mnemonik :

   b). Bahasa mesin :

8. Memindahkan isi Register Port I/O yang alamatnya 00H ke Register A

   a). Mnemonik :

   b). Bahasa mesin

1.2.4 Jawaban 2

1. Mengisi Register B dengan D8H

   a). Mnemonik : LD B,D8H

   b). Bahasa Mesin : 06H D8H

2. Memindahkan isi Register B ke Register H, bila isi Register B = D8H

   a). Mnemonik :LD H,B

   b). Bahasa mesin : 60H

3. Memindahkan isi register A ke lokasi Memori beralamat 0900H

   a). Mnemonik : LD (0900H), A

b). Bahasa mesin : 32H 00H 09H

4. Memindahkan isi lokasi Memori ber alamat 0904H ke Register A

   a). Mnemonik : LD A,(0904)

   b). Bahasa mesin : 3AH 04H 09H

5. Memindahkan isi Register B ke lokasi Memori yang alamatnya ditunjuk oleh Register HL, bila isi Register HL = 0908H

   a). Mnemonik : LD B,(HL)

   b). Bahasa mesin : 46H

6. Memindahkan isi lokasi Memori yang alamatnya ditunjuk oleh Register HL ke register C, bila isi lokasi Memori beralamat 0907H = 09H

   a). Mnemonik : LD (HL), C

   b). Bahasa mesin : 71H

7. Memindahkan isi Register A ke Register Port I/O yang alamatnya 00H

   a). Mnemonik : OUT (00), A

   b). Bahasa mesin : D3H 00H

8. Memindahkan isi Register Port I/O yang alamatnya 00H ke Register A

   a). Mnemonik : LD A,(00)

   b). Bahasa mesin : 3EH 00H

3. INSTRUKSI UNTUK SISTEM KERJA PERINTAH FLAG.

Instruksi ini merupakan instruksi yang diaplikasikan untuk pelaksanaan instruksi lain dengan memanfaatkan kondisi atau status flag, untuk melakukan itu diper-lukan cara untuk menentukan status dari flag tersebut.

Penjelasan berikut tentang bagaimana mengisi Register Flag dan Akkumulator (Program Status Wor), memanfaatkan fungsi Flag zero, memanfaatkan fungsi Flag sign, manfaat fungsi Flag parity dan memanfaatkan fungsi Flag carry.

Flag adalah sebuah flip-flop di dalam blok penghitung dari CPU dan disebut sebagai Register Flag, keadaan Flag ini setelah pelaksanaan sebuah perintah (yang mempengaruhi Flag) akan menghasilkan sifat dari hasil sebuah operasi.

Pada Z-80, Flag di pasangkan dengan Akkumulator dan dikenal dengan program status wort (PSW), berikut struktur bit dari flag:

AKKUMULATOR								FLAG							
D7							D0	D7							D0
								S	Z	-	H	-	P/V	N	C

a. Flag Zero

Itu menunjukkan, apakah pada pelaksanaan terakhir ini operasi hasil pada semua bit adalah = 0.

Kondisi Flag

- **Flag Zero** **= 1,** bila pada semua bit pada Register hasil = 0.
- **Flag Zero** **= 0,** bila semua bit pada Register hasil $\neq$ 0.

Contoh :

```
   0000
   0000  +
 1 0000
```

maka hasil yang didapat pada flag adalah Flag Zero = 1

Flag Carry = 1

b. Flag Carry

Kondisi ini menunjukkan apakah pada proses operasi sebuah bit carry dipindah kan dari bit tertinggi MSB pada Register hasil , itu dapat terjadi pada operasi :

- **Penjumlahan**, bila hasil dari 8 bit atau 16 bit.
- **Pengurangan a - b**, bila b>a, hasil juga negatif.
- **Pergeseran**, bila nilai 1 pada bit tertinggi atau terendah di geserkan ke carry.

Kondisi Flag

- **Flag Carry** = **1**, bila terjadi Carry.
- **Flag Carry** = **0**, bila tidak terjadi Carry.

Flag Carry dapat set melalui perintah SCF dan dibalik melalui perintah SCF.

c. Flag Sign

Pada operasi yang mempengaruhi Flag, Flag sign menyimpan kondisi bit tertinggi dari Register dan hasilnya, menjadi :

Kondisi Flag

- **Flag Sign** = **1**, bila bit tertinggi dari Register hasil = 1
- **Flag Sign** = **0**, bila bit tertinggi dari Register hasil = 0

d. Flag Parity/Overflow

Bit ke 2 dari Register Flag mempunyai 4 arti yang berbeda, tergantung dari hasil akhir pelaksanaan operasi .

- **Flag Overflow**

  Pengertian ini berlaku setelah pelaksanaan perintah berikut :

  - **ADD, ADC, SUB, SBC.**
  - **INC, DEC**

Flag overflow diset 1 pada proses perpindahan dari bit ke 7 ke bit ke 8, yaitu yang mempengaruhi tanda bilangan positif atau negatif pada perhitungan bilangan.

e. Flag Parity

Pengertian ini berlaku setelah pelaksanaan perintah berikut ini :

- Perintah logika **AND, OR, XOR**
- Perintah geser **RL, RR, RLC, RRC**

  SLA, SRA, SRL

  RLD, RRD
- Aritmatik BCD **DAA**
- Perintah input dengan pengalamatan tidak langsung **IN r, (C)**.

Kondisi Flag

- ♦ **Flag Parity = 1**, bila jumlah 1 dan hasil akhir operasi adalah genap
- ♦ **Flag Parity = 0**, bila jumlah 1 dan hasil akhir operasi adalah ganjil

f. Penunjukan Nol Pada Perintah Blok

Pada perintah berikut untuk transfer blok dan pengamatan blok, Flag P?V menunjukan keadaan

Register BC, yang pada operasi ini dipakai sebagai Register penghitung.

- a) Transfer blok **LDI, LDIR, LDD, LDDR**
- b) Pengamatan blok **CPI, CPIR, CPD, CPDR**

Itu berlaku :

- ♦ **Flag P/V = 0,** bila Register penghitung BC = 0000H
- ♦ **Flag Parity = 1,** bila Register penghitung BC ≠ 0000H

g. Penunjukan Dari Pengaktif Flip-Flop Intrupsi (IFF2)

Ini adalah kemungkinan satu-satunya, IFF2, yang menunjukkan keadaan yang sah untuk proses interupt , untuk membaca penggunaan dari Flag P/V ini , berlaku untuk perintah **LD A,I** dan **LD A,R**

1.2.5 Latihan 3

1. Jelaskan apa fungsi dari register flag!
2. Jika hasil matematika pada ALU adalah nol, bagaimana status flag zero?
3. Apa yang ditunjukan flag carry jika data yang dijumlahkan adalah FF dengan 05 ?
4. Jika hasil penjumlah bernilai positip, apa status register flag?
5. Untuk hasil proses memiliki nilai bit ganjil, register flag akan menunjuk apa?

1.2.6 Jawaban 3

1. Flag - Flag pada Register Flag sangat menentukan operasi apa yang dilaksanakan .
2. Maka Flag zero (ZF) = 1
3. Maka Flag carry = 1
4. Maka Flag sign = 1
5. Maka nilai Flag parity =1

3. OPERASI INSTRUKSI ARITMATIKA

Operasi instruksi ini merupakan instruksi matematika yang menjadi dasar perhitungan dalam dunia komputer, sedangkan operasi matematika yang dapat dilakukan meliputi:

⇒ Operasi penjumlahan dan pengurangan dengan operasi 8 bit.

⇒ Operasi penjumlahan dan pengurangan dengan operasi 16 bit

a. Operasi aritmatika dengan operasi 8 bit.

➢ **Register dengan register CPU A, B, C, D, E, F, G, H, L.**

Operasi ini mempunyai format seperti dibawah ini :

ADD : hasil = operan 1 + operan 2

SUB ; hasil = operan 1 - operan 2

ADC : hasil = operan 1 + operan 2 + carry

SBC : hasil operan 1 - operan 2 - carry

Kedua operasi disepakati sebagai bilangan biner dan operasinya berlangsung didalam ALU dan pengalih bilangan setiap instruksi berlaku 2 operan saja.

Mnemonik :

ADD A, r	SUB r	ADC A, r	SBC A, r

Operasi :

Isi dari akku dan register CPU disepakati sebagai bilangan biner 8bit dan saling ditambahkan. Hasil berada pada akkumulator.

<table>
<tr><td>A = A+r</td><td>A = A-r</td><td>A = A+r+C</td><td>A = A-r-C</td></tr>
<tr><td colspan="2">Isi akku yang baru adalah isi akku yang lama + atau isi register CPU r</td><td colspan="2">Pada perintah ini, hasil masih ditambahkan pula isi dari flag carry

Contoh :

Operan 1 80 H
Operan 2 + 20 H
Hasil sementara A0 H
Carry flag + 01 H
Hasil akhir A1 H
Pada akku</td></tr>
</table>

Format

Dalam penjelasan format perintah bentuk ini, kita kelompokkan atas jenis pengalamatan Register dengan regisater CPU A, B, C, D, E, H, L.

1	0	1	0	0	r	r	r	ADD A, r

1	0	0	0	1	r	r	r	SUB r

1	0	0	0	1	r	r	r	ADC A, r

1	0	0	1	1	r	r	r	SBC A,r

r = Pengalamatan register CPU	A = 111	E = 011
	B = 000	H = 100
	C = 011	L = 101
	D = 010	

➢ **Konstanta 8 bit**

Sebagai operan 2 digunakan konstanta yang penulisanya mengikuti Op - Code

1	1	0	0	0	1	1	0	ADD A, n
Konstanta 8 bit								

1	1	0	1	0	1	1	0	SUB n
Konstanta 8 bit								

1	1	0	0	1	1	1	0	ADC A, n
Konstanta 8 bit								

1	1	0	1	1	1	1	0	SBC A, n
Konstanta 8 bit								

- **Register tidak langsung**

Sebagai operan 2 digunakan isi dari lokasi memori yang ditunjukkan melalui register CPU 16 bit HL

1	0	0	0	0	1	1	0	ADD A, (HL)

1	0	0	1	0	1	1	0	SUB (HL)

1	0	0	0	1	1	1	0	ADC A, (HL)

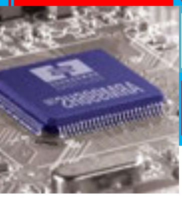
1	0	0	1	1	1	1	0	SBC A, (HL)

- **Indeks offset**

Sebagai operan 2 digunakan isi dari lokasi memori yang ditunjukkan melalui register index + offset e

1	1	0	1	1	1	0	1	
1	0	0	0	0	1	1	0	ADD A,

1	1	1	1	1	1	0	1	
1	0	0	0	0	1	1	0	ADD A, (IY+e)



								(IX+e)
Konstanta offset e								

Konstanta offset e							

1	1	0	1	1	1	0	1	
1	0	0	1	0	1	1	0	SUB (IX+e)
Konstanta offset e								

1	1	1	1	1	1	0	1	
1	0	0	1	0	1	1	0	SUB (IY+e)
Konstanta offset e								

1	1	0	1	1	1	0	1	
1	0	0	0	1	1	1	0	ADC A, (IX+e)
Konstanta offset e								

1	1	1	1	1	1	0	1	
1	0	0	0	1	1	1	0	ADC A, (IY+e)
Konstanta offset e								

1	1	0	1	1	1	0	1	
1	0	0	1	1	1	1	0	SBC A, (IX+e)
Konstanta offset e								

1	1	1	1	1	1	0	1	
1	0	0	1	1	1	1	0	SBC A, (IY+e)
Konstanta offset e								

FLAG

Semua Flag terpengaruh

b. Operasi aritmatika dengan operasi 16 bit

Mnemonik

ADD HL, rr rr = BC, DE, HL, SP	ADD IX, rr rr = BC, DE, SP, IX	ADD IY, rr RR = BC,DE,SP,IY

Operasi

HL = HL + rr IX = IX + rr IY = IY + rr

Isi dari register CPU yang baru (HL, IX atau IY) terdiri dari penjumlahan isi yang lama dengan isi register CPU yang lain.

Format

0	0	r	r	1	0	0	1

rr = Register 16 bit

BC = 00

DE = 01

HL = 10

SP = 11

1	1	0	1	1	1	0	1
0	0	r	r	1	0	0	1

BC = 00

DE = 01

HL = 10

IX = 11

1	1	1	1	1	1	0	1
0	0	R	r	1	0	0	1

BC = 00

DE = 01

HL = 10

IY = 11

Flag

Bit flag H, N dan C terpengaruh.

Juga pada aritmatika dengan operan 16 bit terdapat perintah yang melibatkan bit carry dalam perhitungan.

Mnemonik

ADC HL, rr	SBC HL, rr
rr = register 16 bit CPU	BC, DE, HL, SP

Operasi

HL = HL + rr + C

Isi yang baru dari HL terdiri dari hasil penjumlahan yang lama dari HL ditambah/dikurangi isi dari register 16 bit CPU dan isi bit carry.

Format

1	1	1	0	1	1	0	1
0	1	r	r	1	0	1	0

1	1	1	0	1	1	0	1
0	1	r	r	0	0	1	0

rr = Register 16 bit

BC = 00

DE = 01

HL = 10

SP = 11

Flag :

Semua bit flag terpengaruh.ADD HL, rr

1.2.7 Latihan 4

1. Selesaikan tabel operasi penjumlahan antara isi register dengan isi akkumulator dan hasilnya disimpan diakkumulator.

Mnemonik	Operasi	Op Code	Keterangan
ADD A, B			
ADD A, C			
ADD A, E			
ADD A, H			
ADD A, L			
ADD A, A			

1	0	0	0	0	←	r	→

A = 111 B = 000 C = 001

D = 010 E = 011 H = 100

L = 101

2. Selesaikan tabel operasi pengurangan antara akkumulator dengan register (A - r) dan hasilnya disimpan di akkumulator.

Mnemonik	Operasi	Op Code	Keterangan
SUB B			
SUB C			
SUB D			
SUB E			
SUB H			
SUB L			
SUB A			

1	0	0	1	0	←	r	→

3. Selesaikan tabel operasi penjumlahan antara akkumulator + register + carry dan hasilnya disimpan di akkumulator.

Mnemonik	Operasi	Op Code	Keterangan
ADC A, B			
ADC A, C			
ADC A, E			
ADC A, H			
ADC A, L			
ADC A, A			

1	0	0	0	0	←	r	→

4. Buat program guna menjumlah isi memori alamat 1987H dengan 1978H
   - Isi memori pada alamat 1987H = 78 H
   - Isi memori pada alamat 1978H = 81 H

   Hasil penjumlahan tersebut disimpan diregister H

1.2.8 Jawaban 4

1. Selesaikan tabel operasi penjumlahan antara isi register dengan isi akkumulator dan hasilnya disimpan di akkumulator !

Mnemonik	Operasi	Op Code	Keterangan
ADD A, B	A ← (A) + (B)	80	
ADD A, C	A ← (A) + (C)	81	
ADD A, D	A ← (A) + (D)	82	
ADD A, E	A ← (A) + (E)	83	

ADD A, H	A ← (A) + (H)	84	
ADD A, L	A ← (A) + (L)	85	
ADD A, A	A ← (A) + (A)	87	(A) + (A) =2(A)

1	0	0	0	0	←	r	→

A = 111 B = 000 C = 001

D = 010 E = 011 H = 100

L = 101

2. Selesaikan tabel opersai pengurangan antara akkumulator dengan register (A - r) dan hasilnya disimpan di akkumulator !

Mnemonik	Operasi	Op Code	Keterangan
SUB B	A ← (A) - (B)	90	
SUB C	A ← (A) - (C)	91	
SUB D	A ← (A) - (D)	92	
SUB E	A ← (A) - (E)	93	
SUB H	A ← (A) - (H)	94	
SUB L	A ← (A) - (L)	95	
SUB A	A ← (A) + (A)	97	(A) -(A) = 0

1	0	0	1	0	←	r	→

3. Selesaikan tabel operasi penjumlahan antara akkumulator + register + carry dan hasilnya disimpan di akkumulator !

Mnemonik	Operasi	Op Code	Keterangan
ADC A, B	A ← (A) + (B) + (Cr)	88	
ADC A, C	A ← (A) + (C) + (Cr)	89	
ADC A, D	A ← (A) + (D) + (Cr)	8A	
ADC A, E	A ← (A) + E) + (Cr)	8B	
ADC A, H	A ← (A) + (H) + (Cr)	8C	
ADC A, L	A ← (A) + (L) + (Cr)	8D	
ADC A, A	A ← (A) + (A) + (Cr)	8F	(A) -(A) = 2(A) + (Cr)

1	0	0	0	1	←	R	→

4. Buat program guna menjumlah isi memori ADD 1987 dengan 1978
   - Isi memori pada adress 1987 = 78 H
   - Isi memori pada adress 1978 = 81 H

   Hasil penjumlahan tersebut disimpan diregister H

ADDRESS	B MESIN	B ASSEMBLY	KETERANGAN
1800	3E	LD A, 00H	
1801	00		Data
1802	21	LD HL, 1987H	
1803	87		Data
1804	19		
1805	7E	LD A, (HL)	

1806	21	LD HL, 1978H	
1807	78		
1808	19		Data
1809	46	LD B, (HL)	
180A	80	ADD A, B	Isi reg A + isi reg B
180B	67	LD H, A	
180C	FF	HALT	
1978	81		Data yang akan ditambah
1987	78		Data yang akan ditambah

4. OPERASI PERINTAH BINER

a. Perintah AND

Perintah atau instruksi AND merupakan instruksi logika yang dapat dilakukan dalam ALU, operasi instruksi AND dapat digunakan untuk menghapus bit yang diinginkan dari sebuah data 8 bit.

Contoh :

LD B, C1H	B = 1 1 0 0 0 0 0 1	C1H
IN A, (20H)	A = x x x x xx x x	X = 0 atau 1
AND B	$\wedge$ =______________.	
	A = x x 0 0 0 0 0 x	Hasil

Bit 5 ....1 direset 0, bit yang lainnya yaitu bit 7, bit 6 dan bit 0 tetap tidak berubah yang nilainya tergantung dari nilai x. Jika x=0 maka hasilnya adalah 0 dan jika x=1 maka hasilnya adalah 1.

b. Perintah OR

Perintah atau instruksi OR merupakan instruksi logika yang dapat dilakukan dalam ALU, operasi instruksi OR dapat digunakan untuk digunakan untuk mengeset bit yang dinginkan dari sebuah data 8 bit.

Contoh :

IN A, (20H)	A = x x x x x x x x	x = 0 atau 1
OR 3CH konstanta	= 0 0 1 1 1 1 0 0	3CH
	∨ ________________.	
	A = x x 1 1 1 1 x x	**HASIL**

c. Perintah EX-OR

Perintah atau instruksi EX-OR merupakan instruksi logika yang dapat dilakukan dalam ALU, operasi instruksi EX-OR dapat digunakan untuk membalik bit yang diinginkan dari sebuah data 8 bit.

Dengan perintah ini juga dapat dipakai untuk menyamakan 2 byte, bit-bit yang tidak sama pada akku akan ditunjukkan sebagai 1

Contoh :

IN A, (20H)	A = 1 0 0 0 1 1 1 1	8FH
XOR 3CH konstanta	= 1 1 0 0 0 0 1 1	C3H
	∀ ________________.	
	A = 0 1 0 0 1 1 0 0	**HASIL**

5. OPERASI INSTRUKSI ROTASI/PUTAR

Instruksi rotasi atau geser dan instruksi putar pada umumnya digunakan untuk manipulasi byte pada sebuah register atau memori, instruksi ini juga merupakan instruksi assembler yang ada pada CPU Z-80.

Untuk dapat memahami penggunaannya berikut dijelaskan beberapa instruksi terkait dengan instruksi atau perintah rotasi atau putar pada register, yang

meliputi instruksi rotasi R L r, instruksi rotasi R R r, instruksi rotasi R L C r, instruksi rotasi R R C r, instruksi rotasi S L A r, instruksi rotasi S RL r, dan instruksi rotasi S R A r.

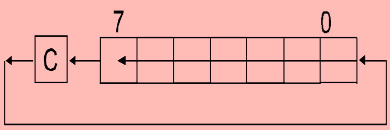
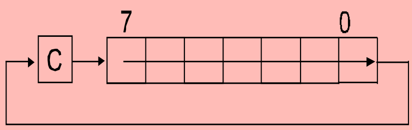
Melalui perintah putar atau geser ini register 8 bit atau lokasi memori dapat digunakan atau difungsikan sebagai sebuah register geser. Operasi instruksi sendiri tentu dilaksanakan dalam ALU yang dapat difungsikan sebagai register-register geser, isi masing-masing bit pada register dapat digeser ke tetangga Flip-flop ke arah kiri atau ke arah kanan.

a. Perintah Putar

➢ **Putar ke kiri atau kanan dalam Register atau memori**

Mnemonik	Mnemonik
R L r	R R r

r = Register CPU 8 bit atau isi memori 8 bit

Operasi :	Operasi :
	
Isi bit 7 dipindahkan ke flag carry dan isi bit Flag Carry dipindahkan ke bit 0 dan bit yang lain digeser 1 ke kiri.	Isi Flag Carry dipindahkan ke bit 7 dan isi bit 0 ke Flag Carry dan bit yang lain digeser 1 kekanan.
Flag :	Flag :
S ●, Z ●, H ○, P/V ●, N ○, C ●	S ●, Z ●, H ○, P/V ●, N ○, C ●

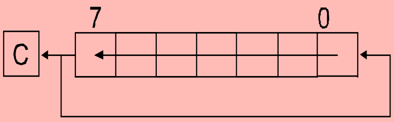
➢ **Putar ke kiri / kanan dalam register atau memori dan menuju flag carry.**

Mnemonik :

RLC r

r = Register CPU 8 bit atau isi memori.

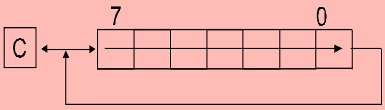
Operasi :



Mnemonik :

RRC r

Operasi :



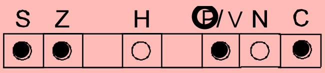
Informasi :

Bit akan bergeser 1 kekiri atau kekanan , tetapi isi flag carry tidak dalam lingkaran perputaran. Bit 7 berpindah ke bit 0 pada RLC, dan bit 0 ber-pindah ke bit 7 pada RRC.

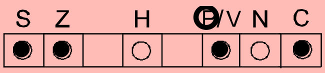
Bit yang lain bergeser satu ke kiri ⟶ RLC.

Bit yang lain bergeser satu ke kanan ⟶ RRC.

Flag :



Flag :



➢ **Putar secara digit ke kiri / kanan**

Mnemonik :

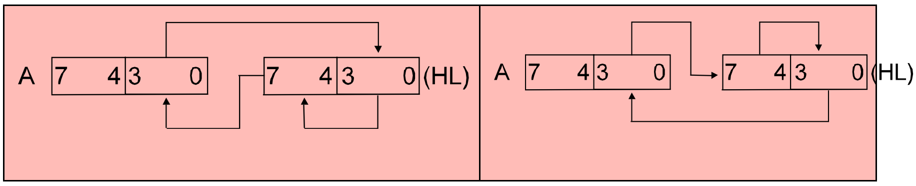
RLD (HL)

Operasi :

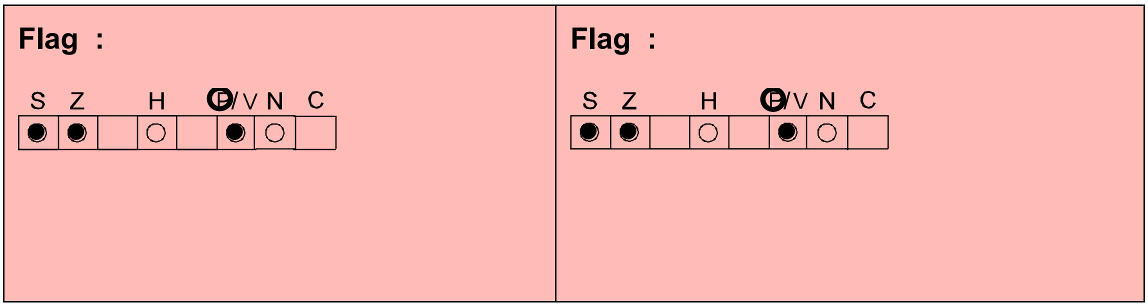
Mnemonik :

RRD (HL)

Operasi :



Ini adalah perintah untuk memutar 12 digit , dan 4 digit untuk pelaksanaan perintah. Posisi perputaran selalu dibagi menjadi separuh dari akkumulator dan sebuah isi memori pengalamatannya melalui (HL). Separuh bagian kiri dari Akku tidak berubah isinya.



b. Perintah Geser

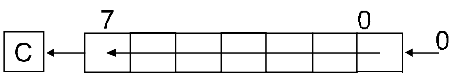
➢ **Geser ke kiri secara aritmetik dalam register atau memori**

Mnemonik :

SLA r

r = Register 8 bit atau isi memori

Operasi :



Informasi

Bit masing-masing digeser 1 (satu) ke kiri, isi bit 7 digeser ke flag carry, pada bit 0 diberikan 0 pada setiap pelaksanaan perintah ini. Bila isi dari register / memori adalah bilangan biner, maka masing-masing bit digeser 1 kekiri. Perintah ini digunakan untuk operasi pengalian.

Flag :

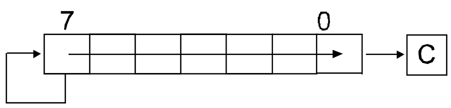
S	Z		H		P/V	N	C
●	●		○		●	○	●

➢ **Geser kekanan secara aritmetik dalam register / memori** :

Mnemonik :

SRA r　　　　　　r = Register 8 bit atau isi memori

Operasi :



Informasi

Bit masing-masing , digeser 1 ke kanan , isi bit 0 digeser ke flag carry , bit 7 selain digeser 1 kekanan , isi bit 7 juga tidak berubah . Perintah ini digunakan untuk operasi pembagian untuk bilangan negatif atau pun bilangan positif.

➢ **Geser ke kanan secara logika dalam register / memori**.

Mnemonik :

SRL r

r = Register 8 bit atau isi memori.

Operasi :

Setiap bit digeser 1 (satu) kekanan, isi bit 0 digeser ke flag carry pada bit 7 diberikan 0 pada setiap pelaksanaan perintah ini.

Flag

S	Z		H		P/V	N	C
●	●		○		●	○	●

Format :

Adr	1	1	0	0	1	0	1	1
adr + 1	0	0	P	P	P	r	r	r

Operasi PPP	Kode Register rrr
RLC = 000	B = 000
RRC = 001	C = 001
RL = 010	D = 010
RR = 011	E = 011
SLA = 100	H = 100
SRA = 101	L = 101
SRL = 111	(HL) = 110
	A = 111

Contoh :

RLC	D	CBH	Memutar ke kiri 1 bit , yang berada pada
		02H	Flag Carry dalam Register D.
RRC	H	CBH	Memutar kekanan 1 bit, yang berada
		0CH	Pada Flag Carry dalam Regiater H.
RL	(HL)	CBH	Memutar ke kiri 1 bit, yang berada pada 16H memori yang ditunjuk oleh HL. bit 7 disalin ke Flag Carry.
SLA	D	CBH	Menggeser ke kiri 1 bit, secara Aritmetik
		22H	isi dari Register.

Flag :

Semua bit Flag terpengaruh.

- **Putar ke kiri / ke kanan dalam memori dengan pengalamatan tidak langsung.**

Mnemonik :

RLD (HL)

Format :

adr	E	D
adr + 1	6	7

Mnemonik :

RRD

Format :

E	D
6	F

Flag : Tidak terpengaruh.

- **Putar ke kiri / ke kanan lewat Carry dalam Akkumulator**

Mnemonik :

RRA

RLA

RRCA

RLCA

Flag : Hanya bit Flag Carry yang berubah.

Format :

untuk RRA, RRA, RLA, RRCA, RLCA

Perintah - perintah ini berlaku untuk mikroprosesor Z80 dan kompatibel dengan instruksi mikroprosesor 8080/8085 dimana Register geser ini hanya bekerja dengan operan yang ada didalam Akku. Kode Operasi perintah ini hanya 1 byte.

RRA	1FH	sesuai dengan	RR A	CBH
				1FH
RLA	17H	sesuai dengan	RL A	CBH
				17H
RRCA	0FH	sesuai dengan	RRC A	CBH
				0FH
RLCA	07H	sesuai dengan	RLC A	CBH
				07H

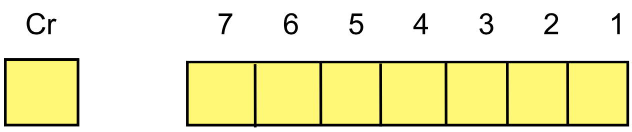
1.2.9 Latihan 5

a. Lengkapilah instruksi rotasi berikut !

1. Rotasi isi akkumulator ke kiri

(RLCA = Rotate left circular Accumulator)

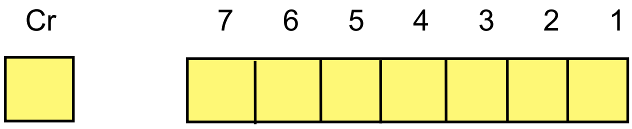
Mnemonic	Operasi	Op code
RLCA	....	07



2. Rotasi isi akkumulator ke kiri melalui carry

(RLA = Rotate left accumulator through carry)

Mnemonic	Operasi	Op code
RLA	....	17



3. Rotasi isi akkumulator kekanan

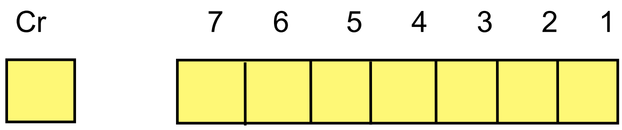
(RRCA = Rotate right circular accumulator)

Mnemonic	Operasi	Op code
RRCA	....	OF

4. Rotasi isi akkumulator ke kanan

(RRA = Rotate right accumulator through carry)

Mnemonic	Operasi	Op code
RRA	....	IF



5. Rotase isi register CPU r ke kiri

(RLC r = rotate register r left circulator)

Mnemonik	Operasi	Opcode	Ket
R L C B	....	CB 00	
R L C C	....	CB 01	
R L C D	....	CB 02	
R L C E	....	CB 03	
R L C H	....	CB 04	
R L C L	....	CB 05	
R L C A	....	CB 07	

6. Rotasi isi register CPU ke kiri melalui carry

(RL r = Rotate left throught carry register)

Mnemonic	Operasi	Opcode	Keterangan
RL B	....		
RL C	....		
RL D	....		
RL E	....		
RL H	....		
RL L	....		
RL A	....		

7. Rotasi isi register CPU ke kanan

(RRC r = Rotate register r right circulator)

Mnemonic	Operasi	Opcode	Keterangan
RRC B	....		
RRC C	....		
RRC D	....		
RRC E	....		
RRC H	....		
RRC L	....		
RRC A	....		

8. Rotase isi register CPU r ke kanan melalui carry

(RR = Rotate right through carry register r)

Mnemonic	Operasi	Opcode	Keterangan
RR B	....		
RR C	....		
RR D	....		
RR E	....		
RR H	....		
RR L	....		
RR A	....		

b. Lengkapilah instruksi geser

1. Geser isi regisrter CPU r aritmetik ke kiri

(SLA r = shift register r left aritmetik)

Mnemonic	Operasi	Opcode	Keterangan
SLA B	....		
SLA C	....		
SLA D	....		
SLA E	....		
SLA H	....		
SLA L	....		
SLA A	....		

2. Geser isi register CPU r logika ke kanan

(SRL r = Shift register r right logical)

Mnemonic	Operasi	Op code	Keterangan
SRL B	....		
SRL C	....		
SRL D	....		
SRL E	....		
SRL H	....		
SRL L	....		
SRL A	....		

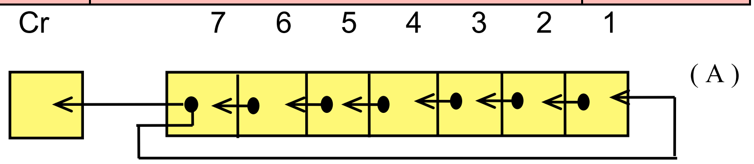
1.2.10 Jawaban 5

a. Lengkapilah instruksi rotasi !

1. Rotasi isi akkumulator ke kiri

   (RLCA = Rotate left circular Accumulator)

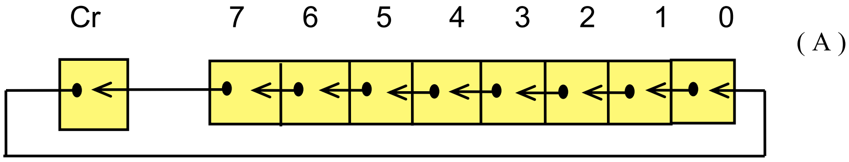
Mnemonic	Operasi	Op code
RLCA	Rotasi isi akkumulator ke kiri	07



2. Rotasi isi akkumulator ke kiri melalui carry

   (RLA = Rotate left accumulator through carry)

Mnemonic	Operasi	Op code
RLA	Rotasi isi akkumulator ke kiri melalui carry	17



3. Rotasi isi akkumulator kekanan

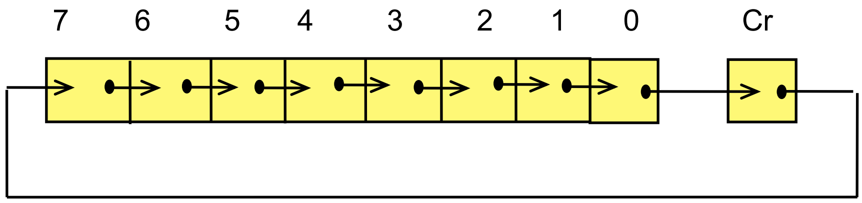
   (RRCA = Rotate right circular accumulator)

Mnemonic	Operasi	Op code
RLA	Rotasi isi akkumulator ke kanan	OF

4. Rotasi isi akkumulator ke kanan

(RRA = Rotate right accumulator through carry)

Mnemonic	Operasi	Op code
RLA	Rotasi isi akkumulator ke kanan melalui carry	IF



5. Rotasi isi register CPU r ke kiri

(RLC r = rotate register r left circulator)

Mnemonik	Operasi	Opcode	Ket
R L C B	Rotasi reg. B ke kiri	CB 00	
R L C C	Rotasi reg. C ke kiri	CB 01	
R L C D	Rotasi reg. D ke kiri	CB 02	
R L C E	Rotasi reg. E ke kiri	CB 03	
R L C H	Rotasi reg. H ke kiri	CB 04	
R L C L	Rotasi reg. L ke kiri	CB 05	
R L C A	Rotasi reg. A ke kiri	CB 07	

6. Rotasi isi register CPU ke kiri melalui carry

(RL r = Rotate left throught carry register)

Mnemonic	Operasi	Opcode	Keterangan
RL B	Rotasi reg. B ke kiri dng Cr	CB 10	
RL C	Rotasi reg. C ke kiri dng Cr	CB 11	

RL D	Rotasi reg. D ke kiri dng Cr	CB 12	
RL E	Rotasi reg. E ke kiri dng Cr	CB 13	
RL H	Rotasi reg. H ke kiri dng Cr	CB 14	
RL L	Rotasi reg. L ke kiri dng Cr	CB 15	
RL A	Rotasi reg. A ke kiri dng Cr	CB 17	

7. Rotasi isi register CPU ke kanan

(RRC r = Rotate register r right circulator)

Mnemonic	Operasi	Opcode	Keterangan
RRC B	Rotasi reg. B ke kanan	CB 08	
RRC C	Rotasi reg. C ke kanan	CB 09	
RRC D	Rotasi reg. D ke kanan	CB 0A	
RRC E	Rotasi reg. E ke kanan	CB 0B	
RRC H	Rotasi reg. H ke kanan	CB 0C	
RRC L	Rotasi reg. L ke kanan	CB 0D	
RRC A	Rotasi reg. A ke kanan	CB 0F	

8. Rotate isi register CPU (r) ke kanan melalui carry

(RR = Rotate right through carry register r)

Mnemonic	Operasi	Opcode	Keterangan
RR B	Rotasi reg. B ke kanan dng Cr	CB 18	
RR C	Rotasi reg. C ke kanan dng Cr	CB 19	
RR D	Rotasi reg. D ke kanan dng Cr	CB 1A	
RR E	Rotasi reg. E ke kanan dng Cr	CB 1B	
RR H	Rotasi reg. H ke kanan dng Cr	CB 1C	

RR L	Rotasi reg. L ke kanan dng Cr	CB 1D	
RR A	Rotasi reg. A ke kanan dng Cr	CB 1F	

b.. Lengkapilah instruksi geser

1. Geser isi regisrter CPU r aritmetik ke kiri

(SLA r = shift register r left aritmetik)

Mnemonic	Operasi	Opcode	Keterangan
SLA B	Geser reg. B aritmetik ke kiri	CB 20	
SLA C	Geser reg C aritmetik ke kiri	CB 21	
SLA D	Geser reg D aritmetik ke kiri	CB 22	
SLA E	Geser reg E aritmetik ke kiri	CB 23	
SLA H	Geser reg H aritmetik ke kiri	CB 24	
SLA L	Geser reg L aritmetik ke kiri	CB 25	
SLA A	Geser reg A aritmetik ke kiri	CB 27	

2. Geser isi register CPU r logika ke kanan

(SRL r = Shift register r right logical)

Mnemonic	Operasi	Op code	Keterangan
SRL B	Geser reg B logika kekanan	CB 38	
SRL C	Geser reg C logika kekanan	CB 39	
SRL D	Geser reg D logika kekanan	CB 3A	
SRL E	Geser reg E logika kekanan	CB 3B	
SRL H	Geser reg H logika kekanan	CB 3C	
SRL L	Geser reg L logika kekanan	CB 3D	

SRL A	Geser reg A logika kekanan	CB 3F	

7 OPERASI INSTRUKSI BIT PADA Z 80

Terdapat 3 (tiga) operasi instruksi bit pada mikroprosesor Z80, dimana masing-masing instruksi terkait dengan penentuan logika pada satu bit isi dari sebuah register.

Instruksi bit ini meliputi instruksi set bit pada suatu register, instruksi Reset bit pada suatu register dan instruksi untuk menguji bit pada register.

Mnemonik :	Operasi :
SET b,r	⇒ Men-Set bit nomor b dari Register / memory, yang dipilih = 1.
RES b,r	⇒ Me-Reset bit nomor b dari Register / memori, yang dipilih = 0.
BIT b,r	⇒ Menguji bit nomor b dari Register / memori, yang dipilih dan berlaku : Bila bit yang diuji = 0, flag zero = 1 Bila bit yang diuji = 1, flag zero = 0

Posisi nomor bit b sebagai operan, mengikuti ketetapan berikut :

7	6	5	4	3	2	1	0

MSB LSB

a. Test Bit b Pada Register r

Mnemonik BIT b,r :

Fungsi : $Z \leftarrow r_b$

Flag : S Z H P/V N C

-

?			1		?	O	

b. Reset Bit b Dari Operan r

Mnemonik RES b,r (Register dan pasangan Register).

Fungsi : $r_b \leftarrow 0$

Flag : S Z H P/V N C

c. Set Bit b Dari Operan r

Mnemonik SET b,r : (Register dan pasangan Register)

Fungsi : $r_b \leftarrow 1$

Flag : S Z H P/V N C

Format :

berlaku untuk :

⇒ Pengalamatan Register

⇒ Pengalamatan tidak langsung melalui (HL)

⇒ Pengalamatan dengan Indeks offset dengan IX atau Iy.

adr	C				B				byte 1
adr + 1	p	p	b	b	b	r	r	r	byte 2

<u>Operasi pp</u>	<u>Bit b b b</u>	<u>Register r r r</u>
BIT = 01	0 = 000	B = 000
RES = 10	1 = 001	C = 001

SET = 11	2 = 010 3 = 011 4 = 100 5 = 101 6 = 110 7 = 111	D = 010 E = 011 H = 100 L = 101 HL = 110 A = 111

1.2.11 Latihan 6

1. Test bit b register r (Bit b,r)

Mnemonic	Operasi	Opcode
BIT b.r	....	op code selalu di mulai dengan "CB"

1	1	0	0	1	0	1	1	Byte 1

0	1		b			r		Byte 2

Reg / Bit	A (111)	B (000)	C (001)	D (010)	E (011)	H (100)	L (101)
O (000)							
1 (001)							
2 (010)							
3 (011)							
4 (100)							
5 (101)							
6 (110)							
7 (111)							

Keterangan :

- Jika bit yang ditunjuk berisi “0 “ , maka Z di set “1”
- Jika bit yang ditunjuk berisi “1”, maka Z diset “0”

2. Set bit b register r (Set b,r)

Mnemonic	Operasi	Opcode
SET b.r	....	op code selalu di mulai dengan “CB”

1	1	0	0	1	0	1	1	Byte 1

0	1		b			r		Byte 2

Reg / Bit	A	B	C	D	E	H	L
0							
1							
2							
3							
4							
5							
6							
7							

3. Reset bit b register r (Res b,r)

Mnemonic	Operasi	Opcode
RES b,r	....	op code selalu di mulai dengan "CB"

1	1	0	0	1	0	1	1	Byte 1

0	1		b			r		Byte 2

Reg / Bit	A	B	C	D	E	H	L
0							
1							
2							
3							
4							
5							
6							
7							

1.2.12 Jawaban 6

1. Test bit b register r (Bit b,r)

Mnemonic	Operasi	Opcode
BIT b.r	Test bit b Register r	op code selalu di mulai dengan "CB"

1	1	0	0	1	0	1	1	Byte 1

0	1		b			r		Byte 2

Reg / Bit	A (111)	B (000)	C (001)	D (010)	E (011)	H (100)	L (101)
O (000)	47	40	41	42	43	44	45
1 (001)	4F	48	49	4A	4B	4C	4D
2 (010)	57	50	51	52	53	54	55
3 (011)	5F	58	59	5A	5B	5C	5D
4 (100)	67	60	61	62	63	64	65
5 (101)	6F	68	69	6A	6B	6C	6D
6 (110)	76	70	71	72	73	74	75
7 (111)	7F	78	79	7A	7B	7C	7D

2. Set bit b register r (Set b,r)

Mnemonic	Operasi	Opcode
SET b.r	Set Bit b Register r	op code selalu di mulai dengan “CB”

1	1	0	0	1	0	1	1	Byte 1

0	1		b			r		Byte 2

Reg / Bit	A	B	C	D	E	H	L
0	C7	CD	C1	C2	C3	C4	C5
1	CF	C8	C9	CA	CB	CC	CD
2	D7	D0	D1	D2	D3	D4	D5
3	DF	D8	D9	DA	DB	DC	DD
4	E7	E0	E1	E2	E3	E4	E5
5	EF	E8	E9	EA	EB	EC	ED
6	F7	F0	F1	F2	F3	F4	F5
7	FF	F8	F9	FA	FB	FC	FD

3. Reset bit b register r (Res b,r)

Mnemonic	Operasi	Opcode
RES b.r	Reset bit b Register r .	op code selalu di mulai dengan “CB”

1	1	0	0	1	0	1	1	Byte 1

0	1		b			r		Byte 2

Reg / Bit	A	B	C	D	E	H	L
O	87	80	81	82	83	84	85
1	8F	88	89	8A	8B	8C	8D
2	97	90	91	92	93	94	95
3	9F	98	99	9A	9B	9C	9D
4	A7	A0	A1	A2	A3	A4	A5
5	AF	A8	A9	AA	AB	AC	AD
6	B7	B0	B1	B2	B3	B4	B5
7	BF	B8	B9	BA	BB	BC	BD

7. OPERASI INSTRUKSI LONCAT

Operasi perintah atau instruksi percabangan memungkinkan dilakukannya pengulangan sebuah bagian program tunggal (penundaan), pemilihan program bagian yang berbeda (keputusan), dan memungkinkan untuk melakukan pengelompokan tugas-tugas yang besar dalam beberapa program-program bagian yang kecil.

Untuk melakukan loncat dalam suatu program percabangan diperlukan instruksi meliputi:

⇒ Perintah percabangan bersyarat dan tak bersyarat

⇒ Perintah loncat dengan pengalamatan langsung

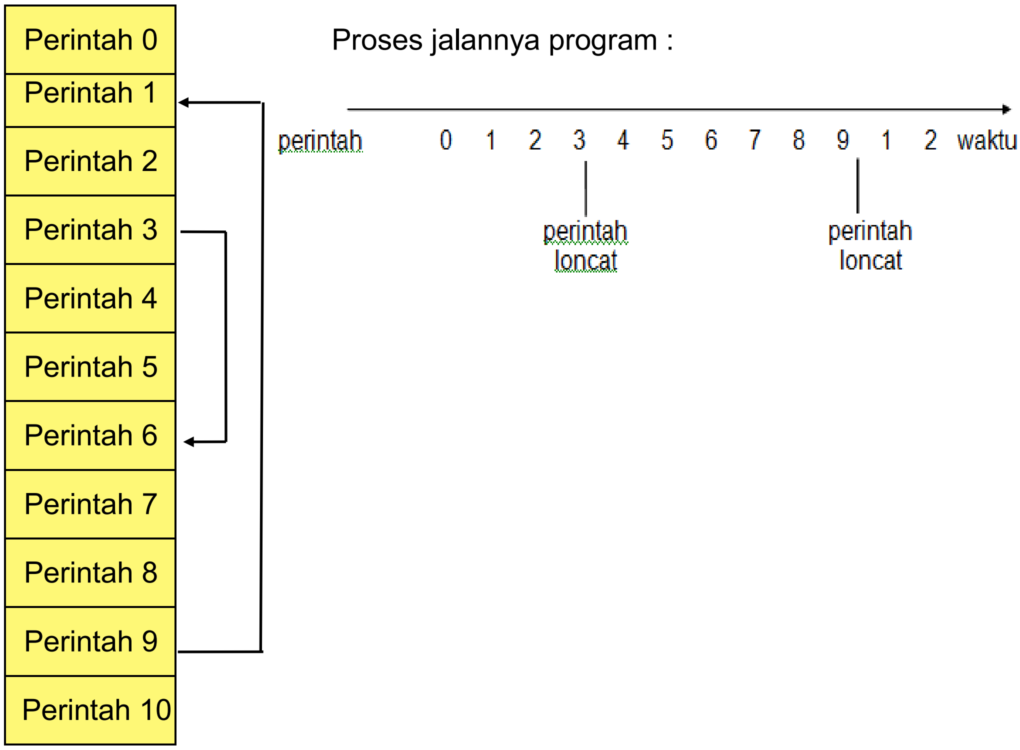
⇒ Perintah loncat dengan pengalamatan relative

⇒ Perintah loncat dengan pengalamatan tidak langsung melalui register

Berdasarkan fungsinya instruksi loncat dapat dikelompokan menjadi beberapa instruksi, yaitu:

- Perintah - perintah loncat (JUMP)
- Pemanggilan program bagian (CALL),
- Loncatan kembali dari program bagian ke program pemanggil (RETURN),
- Perintah start ulang (RST).

Pada pelaksanaan sebuah perintah-perintah percabangan, alamat loncat diikut sertakan ke dalam kode operasinya, dan penunjukan alamatnya akan diisi pada Program Counter (PC). Selanjutnya program akan dijalankan mulai dari alamat memori yang ditunjuk oleh PC, untuk lebih jelasnya perhatikan diagram berikut:



Alamat tujuan loncat yang lebih besar program loncat ke depan, dan untuk alamat tujuan lebih kecil program loncat ke belakang. Alamat tujuan inilah yang merupakan alamat lokasi memori yang disertakan pada perintah percabangan.

a. Perintah Percabangan Bersyarat Dan Tidak Bersyarat

Pada sebuah loncatan tidak bersyarat, pelaksanaan program pada setiap saat dimulai pada alamat tujuan loncat yang telah ditentukan pada perintah percabangan.

Pada sebuah loncatan bersyarat, pelaksanaan program pada alamat tujuan loncat yang telah ditentukan tdilaksanakan bila syarat loncat terpenuhi, bila syarat loncat tidak terpenuhi maka pelaksanaan program dimulai pada perintah berikutnya dibawah perintah percabangan.

Berikut merupakan contoh pelaksanaan instruksi loncat dengan syarat terpenuhi dan belum terpenuhi.

Perinrah		Perinrah
Perinrah		Perinrah
Perinrah		Perinrah
Perinrah		Perinrah
Instruksi loncat ke address 1 jika syarat terpenuhi		Instruksi loncat ke address 1 jika syarat terpenuhi
Perinrah		Perintah
Pelaksanaan loncat dengan syarat terpenuhi		Pelaksanaan loncat syarat belum terpenuhi

Pada perintah percabangan bersyarat, syarat loncat yang diaplikasikan pada operasi instruksi loncat digunakan untuk men-set flag. Berdasarkan kondisi atau status register flag inilah yang digunakan untuk menentukan apakah syarat sudah terpenuhi atau syarat belum terpenuhi, jika terpenuhi maka pelaksanaan

program dipindahkan ke alamat yang ditunjuk oleh PC dan jika belum terpenuhi maka program akan dijalankan terus.

b. Penggunaan perintah percabangan bersyarat

Dalam penggunaan perintah percabangan bersyarat ini, ada 2 hal yang harus jelas yaitu :

– Bagaimana syarat loncat dapat diberikan pada loncatan bersyarat.

– Jika ternyata ada maka diberikan alamat tujuan loncat (pengalamatan).

Syarat loncat / Pengalamatan	Tanpa Syarat	Flag zero	Flag Carry	Flag P/V	Flag S
Langsung	JP adr	JP Z, adr JP NZ, adr	JP C, adr JPNC, adr	JP PE, adr JP PO, adr	JP M, adr JP P, adr
Relatif	JR e	JR Z, e JR NZ, e	JR C, e JR NC, e	-	-
tidak langsung (melalui register)	JP (HL) JP (IX) JP (IY)	-	-	-	-

c. Perintah loncat dengan pengalamatan langsung

Mnemonik :

JP adr

JP cc, adr

adr = alamat tujuan loncat

cc = syarat loncat (kode kondisi flag)

Operasi

PC CPU diisi oleh alamat tujuan yaitu konstanta 16 bit yang mengikuti Opcode. CPU akan	Bila syarat loncat yang ditunjukkan oleh ccc terpenuhi, maka PC akan diisi oleh alamat tujuan yaitu

menjalankan perintah yang ada pada alamat yang ditunjuk oleh PC.	konstanta 16 bit yang mengikuti Op-code, CPU akan menjalankan perintah yang berada pada alamat yang sesuai dengan PC. Bila syarat tidak terpenuhi maka PC = PC + 1, sehingga CPU akan menjalankan perintah yang berada pada alamat sesuai dengan PC.

Format :

Adr	C	3
adr + 1	alamat tujuan LSB	
adr + 2	alamat tujuan MSB	

Op-code	1	1	c	c	c	0	1	0
	alamat tujuan LSB							
	alamat tujuan MSB							

Sebagai syarat loncat hanya dapat dipakai kondisi flag tertentu, seperti ditunjukkan oleh tabel dibawah ini.

Flag	ccc	Mnemonik	Arti
Zero	000	JP NZ, adr	loncat ke alamat tujuan bila hasil <> 0 (Z=0)
	001	JP Z, adr	loncat ke alamat tujuan bila hasil = 0 (Z =1)
Carry	010	JP NC, adr	loncat ke alamat tujuan bila carry = 0 (C=0)
	011	JP C, adr	loncat ke alamat tujuan bila carry = 1 (C=1
P/V	100	JP PO, adr	loncat ke alamat tujuan bila parity ganjil (P/V=0)
	101	JP PE, adr	loncat ke alamat tujuan bila parity genap (P/V=1)
Sign	110	JP P, adr	loncat ke alamat tujuan bila hasil positip (S=0)
	111	JP M, adr	loncat ke alamat tujuan bila hasil negatip (S=1)

Setiap perintah loncat selalu hanya dapat menguji sebuah kondisi flag tertentu yang dihasilkan melalui perintah sebelumnya. Flag Half Carry (H) dan Flag Subtract (N) tidak dapat dipakai sebagai syarat loncat.

Flag : tidak terpengaruh

d. Perintah loncat dengan pengalamatan relatif

Jenis pengalamatan ini, pada Z-80 hanya dipakai untuk perintah loncat.

Mnemonik

JR e

JR cc, e

cc = syarat loncat hanya untuk flag carry dan zero

e = offset, jarak loncat

Format :

Adr	1	8	Op-code	0	0	1	c	c	0	0	0
adr + 1	Konstanta e			Konstanta e							

Pengertian masing-masing bit pada konstanta e, jarak adalah sebagai berikut :

7	6	5	4	3	2	1	0	
V	X	X	X	X	X	X	X	

V = 0 : loncat ke depan 0 0 0 0 0 0 0 0 ... 0 1 1 1 1 1 1 1 → 0 sampai + 127 desimal

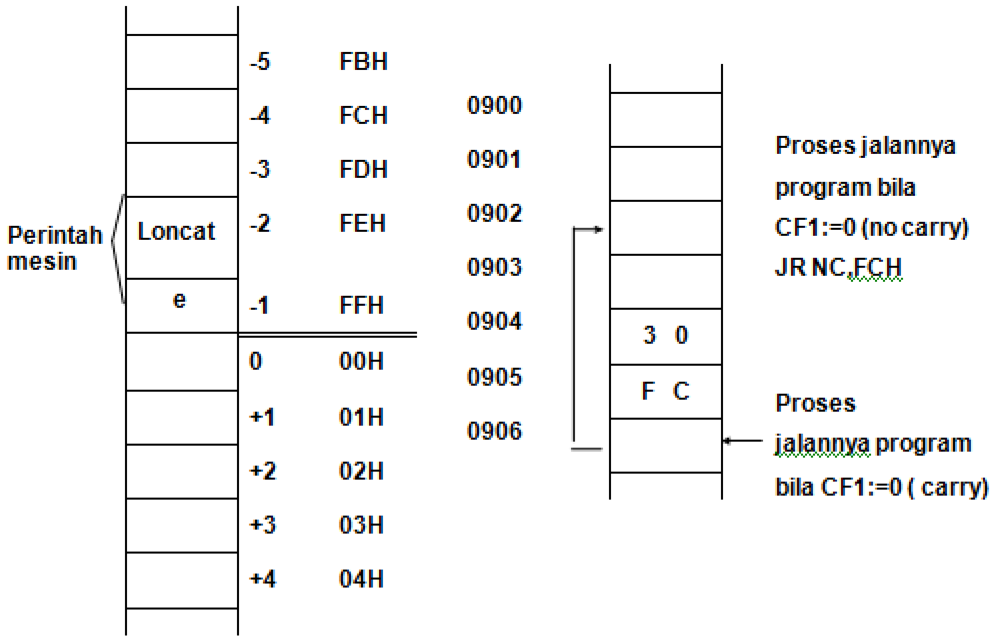
V = 1 : loncat ke belakang 1 1 1 1 1 1 1 1 ... 1 0 0 0 0 0 0 0 → -1 sampai -128 desimal

Operasi

PC CPU diisi oleh alamat tujuan yaitu hasil dari PC saat itu ditambah/dikurangi konstanta e. CPU akan menjalankan perintah pada alamat yang ditunjukkan oleh PC.	Bila syarat loncat yang ditunjukkan oleh cc terpenuhi, maka PC akan diisi oleh hasil penjumlahan/pengurangan PC saat itu dengan konstanta e, bila tidak terpenuhi PC = PC + 1.

Contoh :

JR NC, - 4H	30H FCH	Loncat relatif untuk menjalankan perintah yang berada pada alamat PC -4 (4 alamat ke belakang dari alamat yang sedang ditunjuk PC saat itu), bila pada hasil operasi sebelumnya tidak terjadi carry.



Untuk menghemat pemakaian lokasi memori pada pelaksanaan perintah untuk pembatasan pengulangan dapat dipergunakan.

Mnemonik :

DJNZ e yang fungsinya sama dengan :

DEC B

JR NZ, e yaitu mengurangi 1 dan lompat relatif jika Z = 0

Format :

1	0
Konstanta e	

Flag tidak terpengaruhi

Contoh :

LD HL, 1800 LD B, 00H LD (HL), 00H INC HL DJNZ - 5H	06 00 36 00 23 10 FB	Mengisi Reg B dengan konstanta 00 Mengisi Reg HL dengan konstanta 00 Menambah -1 isi HL Mengurangi -1 isi B dan loncat 5 lokasi kebelakang

e. Perintah loncat pengalamatan tidak langsung melalui register

Disini isi dari register 16 bit dipergunakan sebagai alamat tujuan loncat.

Mnemonik

JP (HL)

JP (IX)

JP (IY)

Operasi :

Meloncat ke alamat lokasi memori yang ditunjuk oleh pasangan register 16 bit.

Isi register tetap tidak berubah.

Flag : tidak terpengaruh

Contoh :

0110 0101 0102 0103	2A 00 0C E9	LD HL,(0C00H) - - JP (HL)	: Alamat tujuan setelah HL diisi. : Loncat ke kelompok yg dipilih.
0300	Kelompok A		
0328	Kelompok B		
034B	Kelompok C		
0C00 0C01	28 03	Pada alamat RAM ini, alamat awal dari kelompok yang terpilih disimpan.	

Loncat dengan pengalamatan tidak langsung melalui register.

1.2.13 Latihan 7

1. Apa kegunaan Perinrah Percabangan ?
2. Sebutkan dua hal penting dalam penggunaan perintah percabangan ?
3. Contoh : <u>Mnemonik</u> : JP adr

   adalah perintah loncat dengan pengalamatan bagaimana ?
4. Contoh : <u>Mnemonik</u> : JP NZ, adr

   apa maksud operasi dari Mnemonik tersebut ?
5. Contoh : <u>Mnemonik</u> : JP (HL)

   adalah perintah loncat dengan pengalamatan bagaimana ?
6. Lengkapilah tabel dibawah ini

Mnemonik	Operasi	Op code	Operand	Keterangan
JPNZ, nn				Not zero (Z=0)
JP Z, nn				Zero (Z=1)
JP NC, nn				No Carry (CY=0)
JP C, nn				Carry (CY=1)
JP PO,nn				Parity odd (P/V=1)
JP PE,nn				Parity even (P/V=1)
JP P,nn				Positive (S=0)
JP M, nn				Minus (S=1)

1	1		cc		0	1	0	Byte 1	NZ = 000; Z = 001
									NC = 010; C = 011
←			n				→	Byte 2 = LOB	PO = 100; PE= 101
									P = 110; M = 111
←			n				→	Byte 3 = HOB	

7. Bila PC berisikan data 2000H dan kita menginginkan PC berisikan FF00H (loncat tak bersyarat), maka perintah loncat apa yang dipergunakan :

   byte 1 = ..., byte 2 = ..., byte 3 = ...,

   byte 1 adalah . . . .

   byte 2 adalah . . . .

   byte 3 adalah . . . .

1.2.14 Jawaban 7

1. Untuk :
   - Pengulangan sebuah bagian program tunggal (penundaan).
   - Pemilihan program bagian yang berbeda (keputusan) dan sebagainya.
2. Dua hal penting yang harus jelas :
   - Bagaimana syarat loncat dapat diberikan pada loncatan bersyarat (syarat loncat)
   - Kemungkinan apa yang ada, untuk memberikan alamat tujuan loncat (pengalamatan)
3. Contoh : Mnemonik : JP adr (alamat)

   Perintah loncat dengan pengalamatan langsung
4. Contoh : Mnemonik : JP NZ, adr(alamat)

   Operasinya : loncat ke alamat tujuan bila Z = 0.
5. Contoh : Mnemonik : JP (HL)

   Perintah loncat dengan pengalamatan tidak langsung → melalui Register.

6. Lengkapilah tabel dibawah ini

Mnemonik	Operasi	Op code	Operand	Keterangan
JP NZ, nn		C2		Not zero (Z=0)
JP Z, nn		CA		Zero (Z=1)
JP NC, nn		D2		No Carry (CY=0)
JP C, nn		DA	LOB	Carry (CY=1)
JP PO,nn		E2	HOB	Parity odd (P/V=1)
JP PE,nn		EA		Parity even (P/V=1)
JP P,nn		F2		Positive (S=0)
JP M, nn		FA		Minus (S=1)

1	1		cc		0	1	0	Byte 1

← n → Byte 2 = LOB

← n → Byte 3 = HOB

NZ = 000; Z = 001

NC = 010; C = 011

PO = 100;PE = 101

P = 110; M = 111

7. Bila PC pada adress 2000 dan kita menginginkan adress PC tersebut, meloncat ke adress FF00 (loncat tak bersyarat), maka :

   byte 1 = C3 . , byte 2 = 00 ., byte 3 = 20

   byte 1 adalah OP CODE

   byte 2 adalah LOB (Lower Order Byte)

   byte 3 adalah HOB (Higher Order Byte

8. OPERASI SUB ROUTINE

Sub routin atau sering disebut sebagai program bagian merupakan program kecil yang dibuat untuk pemakaian yang berulang-ulang, artinya setiap kali ada proses yang sama selalu memanggil program ini.

Fungsi sub routine dalam suatu program memiliki arti yang sangat penting untuk mengurangi sejumlah pemakaian memori, sedangkan perilaku yang harus diberikan untuk memanfaatkan program bagian meliputi:

- Proses pemanggilan sebuah program bagian
- Penyimpan alamat loncat balik pada stack pointer
- Perintah pemanggilan program bagian
- Perintah loncat balik dari program bagian
- Pemberian parameter pada program bagian

a. Program Bagian (Sub Routine)

Dalam program yang mempergunakan sebuah kelompok program yang sering dipakai, maka kelompok program ini dapat ditulis sekali saja, dan dapat dipanggil dimana saja dalam program utama bila kelompok program ini diinginkan.

Kelompok program ini disebut Program Bagian atau Sub Routine

Pemanggilan program bagian dengan mempergunakan perintah CALL (. . . .) atau RST (Restart), selalu menyimpan alamat perintah berikutnya pada Stack, alamat yang disimpan ini menjadi tujuan saat kembali setelah pelaksanaan program bagian.

Perintah terakhir sebuah program bagian adalah selalu sebuah perintah return/kembali (RET), yang fungsinya mengisi penghitung program / program counter (PC), dengan alamat tujuan kembali yang disimpan di stack. Sehingga pelaksanaan perintah berikutnya pada program utama setelah kembali dari program bagian adalah perintah yang alamatnya tersimpan pada .Stack

b. Proses Pemanggilan Sebuah Program Bagian

Pada setiap pemanggilan program bagian dengan CALL, pertama-tama. menyimpan alamat tujuan lompat pada Stack.

Kemudian terjadi sebuah lompatan untuk menjalankan perintah pada alamat pertama dari program bagian (PC = Alamat awal program bagian).

Masing - masing perintah pada program bagian dijalankan secara berurutan.

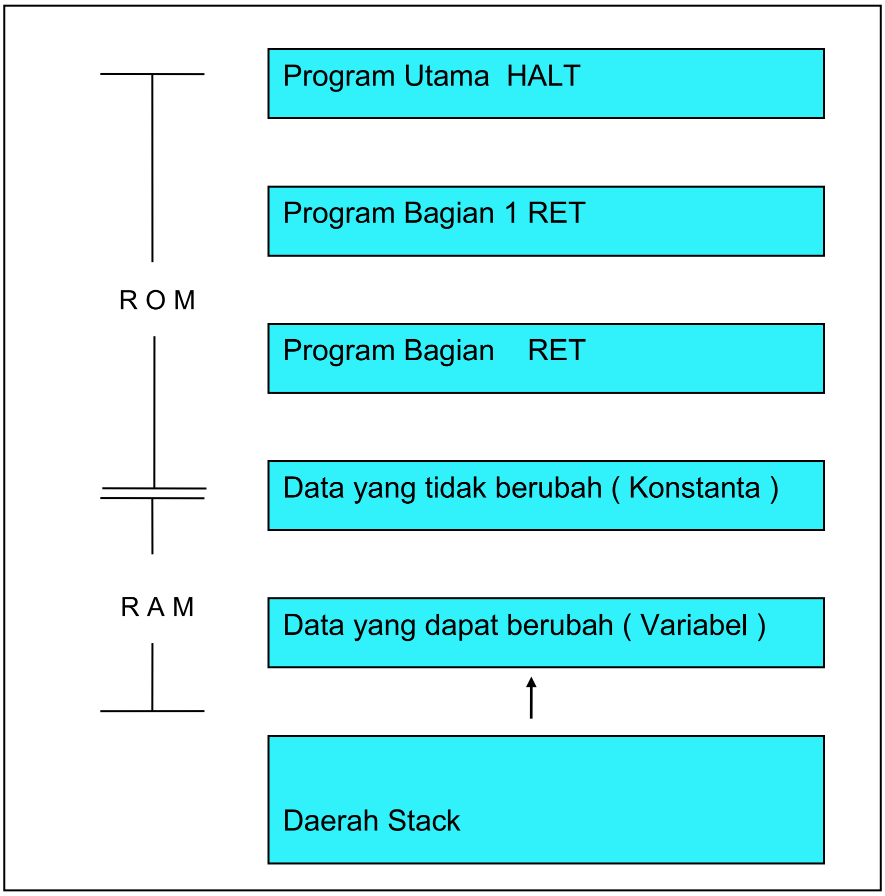
Perintah RET menggunakan alamat loncat balik (dalam Stack), sebagai alamat tujuan loncat balik ke program utama.(PC = alamat loncat balik).

CPU akan melanjutkan pelaksanaan perintah berikutnya pada program utama mulai dari alamat loncat balik ini.

Urutan program bagian adalah bebas dan juga harus tidak saling mengikat dengan program bagian yang lain.

Perintah awal dari program bagian tidak ditentukan secara khusus, hanya perintah akhir dari program bagian harus selalu perintah RETURN (RET).

Peletakan program didalam penyimpan program dan data (memori) lebih banyak seperti pada gambar berikut ini :



c. Penyimpan Alamat Loncat Balik pada STACK

Alamat loncat pada dasarnya diletakkan pada RAM. Penyimpanan di RAM adalah lebih cepat, walaupun demikian kapasitas penyimpannya terbatas.

Untuk penanganan daerah stack, penunjukkan ini dipergunakan fasilitas perintah PUSH dan POP serta register penunjuk STACK (SP).

Pada setiap pemanggilan program bagian , pertama-tama isi Stack Pointer (SP) dikurangi satu (SP - 1) kemudian (SP - 1) diisi dengan MSB (bit tertinggi) dari alamat loncat balik, kemudian isi (SP - 2) diisi dengan LSB (bit terendah) dari alamat loncat balik.

Pada setiap loncat balik dari sebuah program bagian, bagian pertama LSB dari PC akan diisi oleh isi SP (alamat penyimpan stack saat itu) dan bagian MSB dari PC diisi oleh isi (SP + 1) dan kemudian (SP + 2) menunjukkan alamat awal dari pemakaian stack berikutnya.

Pemanggilan program bagian dengan mempergunakan penunjuk stack (SP), memiliki 2 keuntungan, yaitu :

- Pada pemanggilan program bagian yang tidak rumit, dibutuhkan 2 lokasi memori sebagai penyimpan sementara dari alamat loncat balik.
- Program bagian dapat dibuat bercabang secara bebas, yang berarti di dalam sebuah program bagian dapat dibuat program bagian yang lain. Pada setiap pemanggilan program bagian, (SP) akan berubah dan alamat stack yang terakhir akan naik 2 ke atas.

Oleh sebab itu program harus memperhatikan hal itu, yaitu tidak boleh ada tumpang - tindih pada program bagian yang lain, dan tidak melupakan perintah kembali (RET) pada akhir dari masing - masing program bagian.

d. Perintah Pemanggilan Program Bagian

➢ **Perintah Pemanggilan dengan Pengalamatan langsung**

Mnemonik :

CALL adr	CALL cc, adr

adr = Konstanta 16 bit alamat tujuan loncat (alamat pertama dari program bagian)

cc = Syarat loncat/kondisi flag.

Operasi :

pemanggilan program bagian selalu dilaksanakan.	pemanggilan program bagian dijalankan bila syarat CC terpenuhi.

Operasi yang akan dilaksanakan :

– Alamat loncat balik akan disimpan sementara pada stack.

– (SP) diturunkan dua.

– Alamat pertama dari program bagian akan disimpan pada PC.

Format :

adr

C	D
alamat LSB dari prog. bagian	
alamat MSB dari prog. bagian	

<table>
<tr><td>1</td><td>1</td><td>C</td><td>C</td><td>C</td><td>1</td><td>0</td><td>0</td><td></td></tr>
<tr><td colspan="9">alamat LSB dari program bagian</td></tr>
<tr><td colspan="9">alamat MSB dari program bagian</td></tr>
</table>

CCC : 000 ⇒ NZ
001 ⇒ Z
010 ⇒ NC
011 ⇒ C
100 ⇒ PO
101 ⇒ PE
110 ⇒ P
111 ⇒ M

Contoh:

SP : 1FF3H

⇒ Program Utama :

0900	CDH	CALL 0A10H	memanggil program bagian yang ada pada alamat 0A10H
0901	10H	-	
0902	0AH	-	
0903	FFH	HALT	menutup program utama

⇒ Program Bagian :

0A10	3EH	LD A, FFH	mengisi AKKU dengan data FFH
0A11	FFH	-	
0A12	C6H	ADD A, 01H	menambah isi AKKU dengan konstanta 01H
0A13	01H	-	
0A14	C9H	RET	menutup program bagian

⇒ Setelah pelaksanaan program, penunjuk stack (SP) = 1FF3H

1FF1	03H	alamat stack yang akhir
1FF2	09H	
1FF3	XXH	

➢ **Perintah Pemanggilan dengan Perintah Restart/start ulang**

Mnemonik :

RST a

RST : Restart/start ulang pada alamat yang telah ditentukan.

a : 0, 1, 2, 3, 4, 5, 6, 7 atau 00, 08, 10, 18, 20, 28, 30, 38.

Operasi :

Perintah restart adalah pemanggilan program bagian tak bersyarat, yang selalu dilakukan seperti perintah CALL. Operasi - operasi berikut ini akan dilaksanakan :

= Alamat loncat balik akan disimpan sementara pada stack.

= Stack pointer (SP) diturunkan 2.

= Alamat pertama dari program bagian akan disimpan pada PC.

Format :

adr

1	1	a	a	a	1	1	1

aaa : kependekan dari alamat awal program bagian.

Mnemonik Perintah	Op - Code	aaa	Alamat Awal Program Bagian
RST 0 atau RST 00	C7	000	0000H
RST 1 atau RST 08	CF	001	0008H
RST 2 atau RST 10	D7	010	0010H
RST 3 atau RST 18	DF	011	0018H
RST 4 atau RST 20	E7	100	0020H
RST 5 atau RST 28	EF	101	0028H
RST 6 atau RST 30	F7	110	0030H
RST 7 atau RST 38	FF	111	0038H

Perintah RST kebanyakan dipergunakan pada proses pelaksanaan program Interrupt, tetapi dapat juga dipakai pada program normal, yaitu memanfaatkan keuntungan dari formatnya yang rendah, karena kebutuhan penyimpan sementara (RAM) yang sedikit dan kecepatan pelaksanaannya yang cepat bila dibandingkan dengan perintah CALL.

Flag : Flag terpengaruh

Contoh :

⇒ Program Utama :

0100 0101	DFH FFH	RST 3 HALT	memanggil program bagian yang ada pada alamat 0018 H

⇒ Program Bagian :

0018	3EH	LD A, AAH	mengisi AKKU dengan konstanta AAH
	AAH	-	
	C6H	ADD A, 01H	menambah isi AKKU dengan konstanta 01H
	01H	-	
	C9H	RET	menutup program bagian

⇒ Setelah pelaksanaan program, penunjuk stack (SP) = 1FFDH

1FFD	02H	alamat stack yang akhir
1FFE	01H	
1FFF	XXH	

e. Perintah Loncat Balik dari Program Bagian

➢ **Perintah Loncat Balik dari Program Bagian Normal**

Mnemonik :

RET

RET CC

RET = Return, loncat dari program bagian

CC = Syarat loncat balik kondisi flag

Operasi :

- loncat balik tanpa syarat
- loncat balik akan dilaksanakan bila CC terpenuhi.

Setelah pelaksanaan perintah ini, mekanisme pengontrolan akan melaksa-nakan kegiatan berikut ini :

- PC diisi oleh isi dari 2 alamat teratas dari stack.
- SP (Stack Pointer) dinaikkan 2.
- Proses program berikutnya mulai dari alamat yang ada dalam PC.

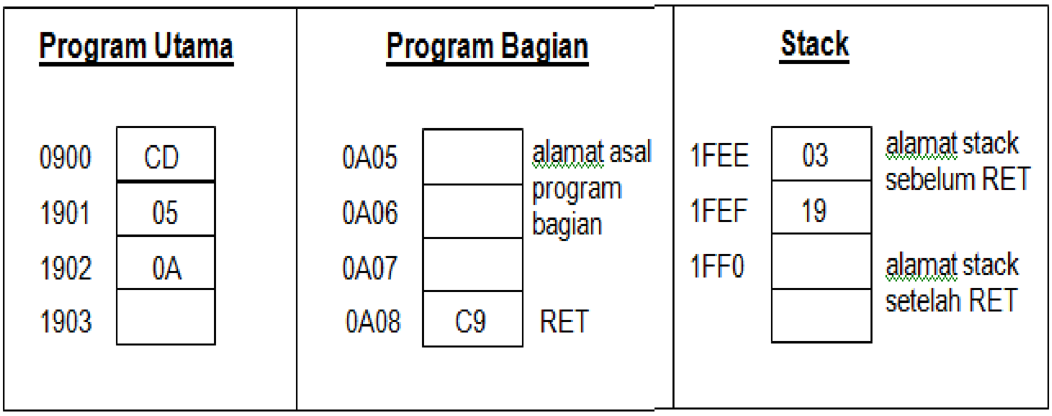
adr		alamat stack sebelum pelaksanaan perintah RET
adr + 1		
adr + 2		alamat stack setelah pelaksanaan perintah RET

Flag :

Flag terpengaruh

Contoh :

SP = 1FF0H



➢ **Perintah Loncat Balik dari Program Bagian**

Perintah ini merupakan perintah yang dipakai untuk loncat balik dari program bagian ke program utama, yaitu yang pemanggilannya melalui pelaksanaan perintah Interupt. Disebut juga sebagai Interupt service routine.

Mnemonik :

RET I	kembali dari maskable Interupt
RET N	kembali dari non maskable interupt

Operasi :

Berjalan dengan logika yang sama seperti perintah RET, khusus untuk keluarga Z 80 yang memiliki blok input - output (I/0). Kode Operasi ini dapat diidentifikasikan (bila kode operasi ini muncul pada BUS data), maka dengan itu CPU dapat menentukan tingkat prioritas pelaksanaan perintah diatas.

f. Pemberian Parameter pada Program Bagian

Nilai masukan, alamat, hasil dan sebagainya, yang dipakai selama pelaksanaan program bagian disebut sebagai PARAMETER. Pemograman harus mempertimbangkan dengan baik, dimana nilai parameter ini disimpan sementara. Program bagian harus ditulis seumum mungkin dan tidak dipakai hanya untuk satu nilai parameter saja.

Juga hasil yang dihasailkan oleh program bagian harus juga diletakkan sedemikian rupa sehingga program utama dapat memberikan nilai yang lain, dengan kata lain, sebuah program bagian selama pelaksanaannya, harus dapat mengalamatkan parameter - parameter yang digunakan.

- **Pemberian Parameter dalam Register CPU**

Parameter yang diperlukan dalam program bagian (yang harus ada sebelum pemanggilan program bagian) di simpan di Register CPU. Ini adalah metode yang sederhana dan cepat, namun hanya dapat digunakan bila terdapat hanya sedikit parameter yang digunakan dan di dalam program utama register tidak dipakai untuk tujuan lain.

Contoh :

Disini sebuah parameter dalam AKKU diberikan ke program bagian .

⇒ Program Utama :

0800	IN A, (20H)	membaca sebuah byte dari port beralamat 20H
....	.....	....
081F	CALL 0A00H	memanggil program bagian
08FF	HALT	penutup program utama

⇒ Program Bagian :

0A00	BIT 2,A	program bagian untuk mengelola data pada AKU
....	....	....
0A5F	RET	penutup program bagian



DASAR PEMROGRAMAN MIKROPROSESOR

DESKRIPSI MATERI PEMBELAJARAN

Berbagai program aplikasi dapat diterapkan pada sebuah *hardware* mikroprosesor, sedangkan program aplikasi didesain sesuai dengan kebutuhan pengembangnya. Sebuah program aplikasi dikembangkan dengan cara menyusun berbagai instruksi-instruksi, dan melalui instruksi inilah komponen-komponen sebuah mikroprosesor dikendalikan. Program aplikasi pada mikroprosesor digunakan untuk memenuhi keperluan dari kendali peralatan rumah tangga, sistem keamanan mobil dan rumah sampai pada sistem kendali di bidang militer dan bidang industri. Untuk membangun sebuah program aplikasi diperlukan sebuah bahasa pemrograman yang disebut dengan bahasa assembly, melalui bahasa ini instruksi disusun dalam urutan sekuensial . Sedangkan penyusunan instruksi tersebut harus mengikuti tata aturan dan konsep yang terstruktur, untuk itu diperlukan konsep algoritma pemrograman yang dapat dituangkan dalam bentuk diagram alir atau struktogram.

KOMPETENSI INTI (KI-3)	KOMPETENSI INTI (KI-4)
Kompetensi Dasar (KD): 1. Mengkonsepkan algoritma dan dia-gram alir pemrogra-man	**Kompetensi Dasar (KD):** 1. Menerapkan algoritma pemro-graman dan diagram alir pemrogra-man
Indikator: 1.1. Memahami pengertian simbol-algoritma dan mengaplikasikan kedalam bentuk instruksi pemrograman 1.2. Memahami diagram alir pemro-graman	***Indikator:*** 1.1. Merencanakan (mengkonsepkan) algoritma dan mendiagramkan diagram alir secara manual *1.2.* Merencanakan (mengkonsepkan) algoritma dan mendiagramkan diagram alir menggunakan bantuan perangkat lunak.

KATA KUNCI PENTING

- instruksi, bahasa assembly
- algortima
- Flowchart, Struktogram

BAB II. DASAR PEMROGRAMAN MIKROPROSESOR

2.1. ALGORITMA PEMROGRAMAN MIKROPROSESOR

Logika akal pikiran manusia terbentuk berdasarkan suatu keinginan, suatu kebutuhan untuk menopang kehidupan yang lebih baik, hal tersebut merupakan tahapan yang selalu dilakukan oleh setiap orang. Untuk mncapai apa yang diinginkan atau untuk memenuhi kebutuhan yang diharapkan hampir selalu dilakukan oleh setiap orang melalui suatu perencanaan berbasis pada logika akal pikiran.

Sebagai contoh seorang sopir akan melakukan perjalanan dari kota Malang menuju kota Surabaya, maka sopir tersebut akan menggunakan akal pikiran dan logikanya mulai dari persiapan segala kelengkapan yag harus disiapkan sebelum berangkat. Mulai berangkat dari mana dan jam berapa, siapa yang ikut serta dan barang bawaan apa saja yang harus dibawa. Rencana jalan yang akan dilalui untuk bisa sampai di tempat tujuan, istirahat dimana berapa lama serta apa yang akan dilakukan di tempat istirahat nanti, demikian seterusnya sehingga sopir tersebut memiliki algoritma berupa rancangan untuk bisa sampai di tempat tujuan.

Bagaimana proses dan dari mana sumber yang dapat dimanfaatkan dalam menyusun rencana dan program pada mikroprosesor menjadi pertanyaan yang sangat penting, hanya dengan logika akal pikiran manusia dapat mewujutkan sebuah rancangan program yang terstruktur, jelas, logis dan mempunyai taret dan tujuan yang jelas. Hampir tidak ada batasan manusia dalam memnfaatkan logika dan akal pikirannya untuk mengkreasi sesuatu yang bermanfaat bagi dirinya maupun orang lain. Seperti diketahui logika dan akal pikiran manusia mampu bekerja selama 24 jam terus menerus, dari kenyataan tentang kerja logika dan akal pikiran manusia ini maka sudah seharusnya kita bersyukur dan mau mengakui keagunganNYA

Identik dengan penjelasan di atas adalah pada sistem mikroprosesor, untuk dapat digunakan sebagai alat bantu dalam menyelesaikan masalah harus diprogram dengan suatu alur penyelesaian masalah yang dirancang sebelumnya.

1. Pengertian Algoritma.

Berbasis pada sebuah buku berjudul Logika Al Jabar Wal Muqabala yang ditulis oleh seorang ilmuwan Arab bernama Abu Jafar Muhammad Ibnu Musa Al Khuwarizmi, membuat penulis tersebut jadi dikenal di seluruh dunia. Nama penulis tersebut Al Khuwarizmi oleh para ilmuwan barat dibaca dengan sebutan Algorism, dan saat ini dikenal dengan sebutan Algorithm.

Algoritma sebutan di bahasa indonesia untuk Algorithm merupakan sebuah ilmu yang terkait dengan penyelesaian masalah, ilmu ini berbasis pada logika pikir untuk melakukan solusi terhadap suatu permasalahan melalui tahapan solusi sekuensial. Sehingga dapat diambil pengertian Algoritma adalah tahapan atau urutan langkah-langkah yang didalamnya berisi solusi logis penyelesaian masalah dan tersusun secara sistematis sehingga dapat mencapai tujuan yang diharapkan.

Contoh permasalahan:

> Si Ali sedang mengikuti rapat di kantor, oleh karena satu dokumen ketinggalan di rumah maka ia memutuskan menelpon adiknya yang berada di rumah untuk mengantarkan dokumen tersebut. Buatlah algoritma yang harus dilakukan oleh Ali sehingga dapat berkomunikasi dengan adiknya agar bisa mengantarkan dokumen tersebut!.

Sebuah alternatip alagoritma dari permasalahan tersebut adalah sebagai berikut:

- Si Ali mengambil HPnya yang berada di dalam tasnya.
- Si Ali meminta ijin kepada pimpinannya untuk ke luar ruangan rapat
- Jika diijinkan maka Ali keluar ruangan maka Ali menelpon adiknya dan minta tolong agar dokumennya diantar ke kantor Ali.
- Jika tidak diijinkan maka Ali tetap mengikuti rapat sampai selesai .
- Ali menunggu kedatangan adiknya yang mengantar dokumen
- Jika adiknya datang membawa dokumen maka dokumen diterima selanjutnya masuk ruangan rapat kembali untuk mengikuti rapat sampai selesai.

- Jika adiknya datang tanpa membawa dokumen maka Ali masuk ke ruangan rapat kembali untuk mengikuti rapatt sampai selesai.

Contoh permasalahan lain:

Ibu membuatkan segelas susu untuk adik dan membuatkan segelas kopi untuk kakak, ternyata ibu dalam membuat kedua minuman tersebut gelas kakak tertukar dengan gelas adik. Kalau gelas tersebut tidak ditukar adik akan menangis karena gelasnya digunakan kakak. Buatlah algoritma yang harus dilakukan oleh ibu agar gelas adik dan gelas kakak saling ditukarkan!

Alternatip algoritma untuk permasalah tersebut adalah:

- Ibu mengambil gelas ke 3 yang digunakan sebagai gelas perantara
- Ibu menuangkan susu dari gelas kakak ke dalam gelas perantara sampai bersih.
- Ibu menuangkan kopi dari gelas adik ke dalam gelas kakak sampai bersih
- Ibu menuangkan susu dari gelas perantara ke dalam gelas adik sampai bersih
- Ibu memberikan gelas adik yang sudah berisi susu kepada adik dan memberikan gelas kakak yang sudah berisi kopi kepada kakak.

2. Kriteria ALgoritma

Terdapat 5(lima) kriteria untuk algoritma dapat dikatakan baik (knuth), yaitu meliputi:

a. Masukan (input), tanpa ada input maka proses dalam algoritma tidak akan jalan, artinya sebuah algoritma harus memiliki data untuk sebuah proses sebagai masukan.
b. Keluaran (Output), ditinjau dari pengertian algoritma di atas maka sebuah algoritma harus memiliki minimal keluaran sebagai tujuan, tanpa ada keluaran yang pasti tujuan dari algoritma tidak bisa dicapai untuk digunakan sebagai solusi permasalahan.



c. Batasan, karena orientasi algoritma pada tujuan maka harus ada batasan dalam mencari solusi, jika hal ini diabaikan maka algoritma akan berpeluang untuk tidak mencapai pada tujuan berupa keluaran yang diinginkan. Untuk algoritma terbuka artinya tanpa adanya batasan maka akan tampak berupa algoritma yang tidak logis dan tidak sistematis serta sulit untuk dimengerti.

d. Arah Tujuan, algoritma harus memiliki kejelasan arah guna mencapai tujuan yang diinginkan, arah dari sebuah algoritma harus diawali dan harus diakhiri, memiliki kejelasan logika dan tahapan sekuensial sehingga didapatkan sebuah hasil berupa keluaran algoritma terkait dengan tujuan.

e. Efisiensi, algoritma harus memiliki efisiensi yang tinggi, artinya tidak merupakan penyelesaian yang abstrak (imajiner) atau bukan angan-angan tetapi lebih merupakan solusi riil guna menyelesaikan suatu masalah. Disini mengandung pengertian untuk tidak melakukan hal-hal yang tidak perlu atau cara memutar-mutar tanpa mengarah pada keluaran sebagai tujuan yang diharapkan.

3. Struktur Algoritma

Dalam menyelesaikan masalah manusia sebagai mahluk Alloh memiliki akal dan pikiran logis, maka suatu algoritma sebagai bentuk penyelesaian masalah akan mengikuti pola akal pikiran manusia. Terdapat 3(tiga) tiga struktur dasar yang dapat dijadikan sebagai acuan pemecahan masalah, yaitu:

a. Penyelesaian bertahap

Penyelesaian bertahap adalah alur pemikiran yang tersuktur, sekuensial, terarah, dan jelas untuk menyelesaikan masalah. Urutan atau sekuensial kegiatan pemecahan masalah dilakukan secara bertahap, dimana setiap tahap akan memberikan hasil dan hasil tersebut akan digunakan untuk proses pada tahap berikutnya. Sebagai catatan bahwa instruksi atau operasi yang digunakan dalam setiap tahap sangat menentukan hasil akhir dari suatu algoritma. Artinya bila tahapan operasi berubah-ubah tidak

konsisten terhadap permasalahan maka mungkin besar hasil yang diaharapkan pada akhirnya tidak akan sesuai dengan tujuan.

b. Pemilihan Alternatip

Pemilihan alternatip yaitu sebuah pilihan yang harus ditentukan pada suatu kondisi tertentu, misal berangkat atau tidak berangkat, dikerjakan atau tidak dikerjakan yang dikerjakan, jika pilih satu maka harus masuk, jika pilih dua maka harus keluar dan jika pilih tiga maka harus tidak berbuat apa-apa dsb. Yang dimaksudkan engan kondisi pada pilihan alternatip adalah persyaratan yang dapat bernilai benar atau salah atau berupa pilihan satu dari sekian alternatip pilihan. Jika operasi kegiatan merupakan pilihan kondisi bernilai benar dan salah maka pernyataaan kondisi menggunakan If dan Then.

c. Proses Pengulangan

Proses Pengulangan adalah operasi kegiatan yang memerlukan tahapan sekuensial berkelanjutan seperti halnya pada penyelesaian bertahap, akan tetapi karena prosesnya sama maka dilakukan kegiatan mengulang sederetan penyelesaian masalah. Kegiatan yang dilakukan secara berulang-ulang tersebut sudah barang tentu disesuaikan dengan persyaratan yang telah ditentukan sebelumnya, dengan demikian tetap akan diperoleh hasil sesuai yang diharapkan. Pernyataan yang digunakan For To Next, Do While, Repeat Until dst.

4. Penulisan Algoritma

Untuk menyajikan Algoritma terdapat kesepakatan dalam bidang pemrograman peralatan elektronik, yaitu dapat dinyatakan dalam bentuk berikut:

- Menggunakan bahasa natural dalam bentuk narasi seperti contoh di atas.
- Menggunakan diagram alir (flow chart)
- Menggunakan Struktogram

Berikut merupakan diskripsi dan contoh-contoh dari ketiga cara penyajian algoritma:

1. Penulisan Algoritma Menggunakan Bahasa Natural

Untuk penyajian algoritma menggunakan bahasa natural dalam bentuk narasi mengacu pada contoh-contoh dan uraian di atas.

2. Diagram Alir (Flow Chart)

Flowchart adalah algoritma penyelesaian suatu masalah yang diwujudkan dalam bentuk penggambaran bagan, dimana dalam bagan tersebut memiliki kandungan aliran data yang lebih menggambarkan langkah-langkah penyelesaian suatu masalah. Terdapat 2(dua) penggambaran Flowchart yaitu System Flowchart dan Program Flowchart.

Flowchart dapat digunakan untuk menggambarkan perilaku suatu algoritma (dengan menggunakan gambar-gambar atau tanda-tanda yang sesuai) Bila suatu *flowchart* lengkap telah selesai dikerjakan, gambaran lengkap tentang proses pemikiran seorang *programmer* dalam memecahkan suatu masalah dapatlah diikuti. Peranan *flowchart* sangat penting terutama pada pemeriksaan program. Flowchart yang merupakan bagian penting dalam suatu program yang telah selesai juga dapat membantu orang lain dalam memahami algoritma yang tepat yang dibuat *programmer.*

Ada dua jenis flowchart, yaitu:

- flowchart sistem: menunjukkan jalannya program secara umum.
- flowchart terperinci: rincian (detail) yang dibutuhkan programer.

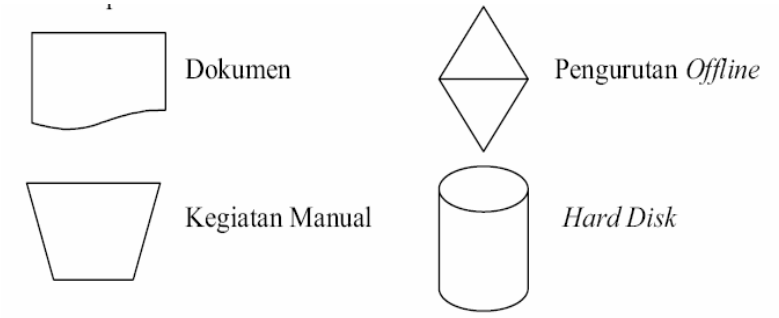
Biasanya suatu program yang rumit didahului dengan flowchart sistem, lalu dilengkapi pula dengan *flowchart* terperinci. Keuntungan dari flowchart ialah bahwa dia menunjukkan urutan langkah-langkah dengan *menggunakan simbol anak* panah.

a. Flowchart Sistem

digunakan untuk menggambarkan aliran data atau informasi secara garis besarnya yang melewati suatu tahapan proses dalam sebuah sistem dengan

menunjukkan media yang digunakan dalam sistem. Seperti media input, output maupun media penyimpanan dalam proses pengolahan data.

Adapun simbol yang digunakan meliputi:



Gambar 5.1. Simbol flowchart sistem

b, Program Flowchart

Digunakan untuk menggambarkan aliran data atau informasi secara rinci yang melewati suatu tahapan proses dengan menunjukkan tahapan penyelesaian permasalahan untuk mendapatkan hasil sesuai tujuan yang diharapkan.

Adapun simbol yang digunakan adalah sebagai berikut:

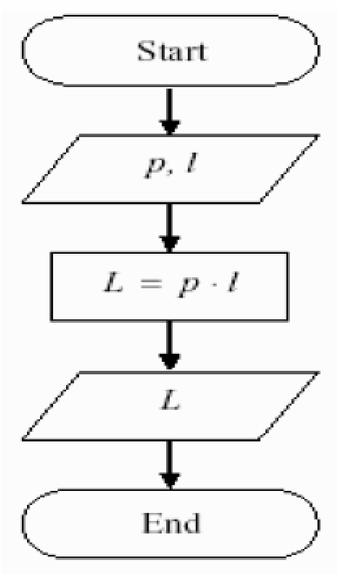
SIMBOL	KETERANGAN
	Terminator
	Flow Line
	Proses
	Input/Output Data
	Decision

SIMBOL	KETERANGAN
	Preparation
	Predefined Process (subprogram)
	On Page Connector
	Off Page Connector

Gambar 5.2. Simbol flowchart umum

Sebuah flowchart minimal umumnya terdiiri dari simbol mulai yang menggambarkan bahwa sebuah algoritma dimulai, dilanjutkan dengan masukan data, selanjutnya data diproses kemudian hasil proses dikeluarkan dan diakhiri dengan simbol akhir dari aliran program.

Berikut contoh flowchart sederhana penyelesaian bertahap untuk menghitung luas sebuah bangun segi empat, dimana bangun tersebut memiki panjang = p dan lebar = l, dimana luas = p x l :

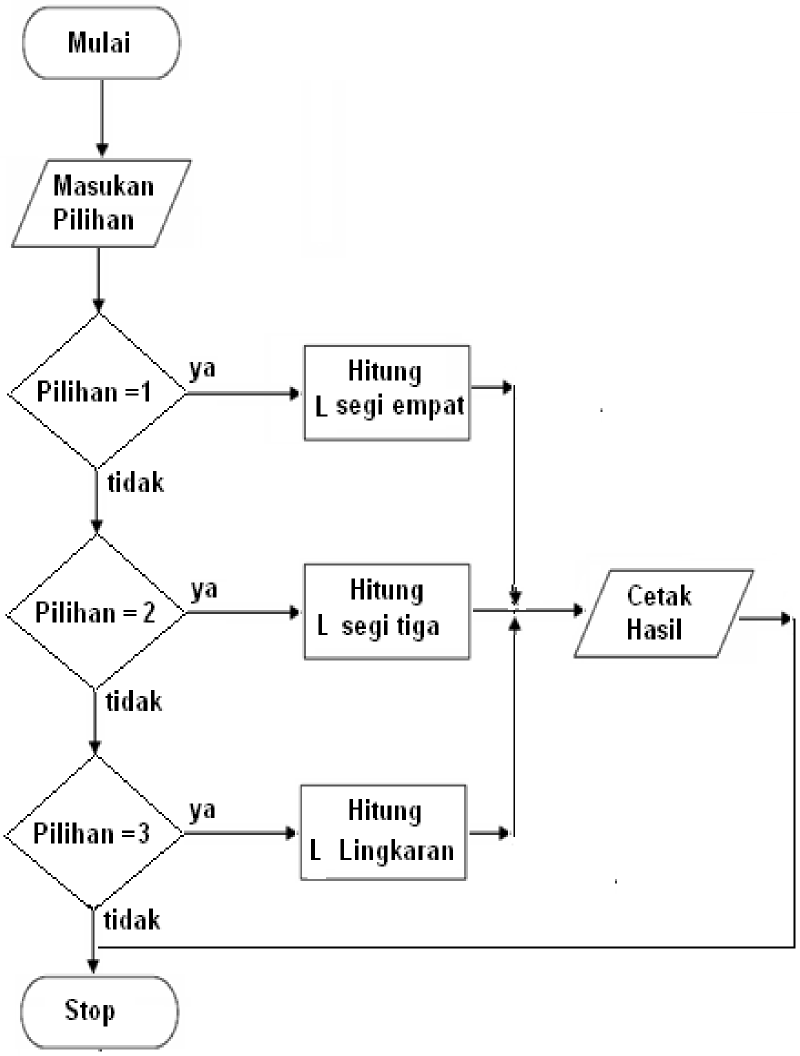


Gambar 5.3. Flowchart sederhana (menghitung luas)

Menghitung luas bangun ternyata tidak hanya sebuah bangun yang bisa dilakukan dalam program, dengan menambahkan menu pilihan ternyata dapat dipilih untuk menghitung beberapa luas bangun dari bidang tertentu.

Jika sebuah algoritma membutuhkan alternatip pilihan maka dibutuhkan simbol percabangan, sebagai contoh sebuah sistem program harus melakukan layanan perhitungan luas bangun segi empat, luas bangun segi tiga dan luas bangun lingkaran.

Dengan demikian maka terdapat pilihan alternatip, jika dipilih 1 maka digunakan untuk menghitung luas bidang segi empat, untuk pilihan 2 digunakan menghitung luas bangun segi tiga dan untuk pilihan 3 digunakan untuk menghitung luas bangun lingkaran, maka flowchartnya dapat digambarkan sebagai berikut:



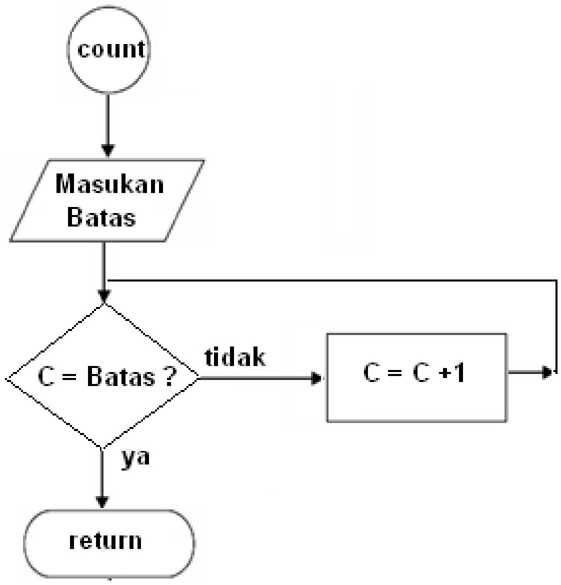
Gambar 5.4 Simbol flowchart sistem

Aliran program dimulai, kemudian memberi kesempatan pada user untuk memberikan masukan berupa angka 1 atau angka 2 atau angka 3, jika angka tersebut adalah 1 maka akan diteruskan pada proses menghitung luas bangun segi empat.

Jika ternyata angka yang diberikan adalah 2 maka akan diteruskan pada proses menghitung luas bangun segi tiga, dan juika ternyata angka yang diberikan adalah 3 maka akan diteruskan pada proses menghitung luas bangun lingkaran.

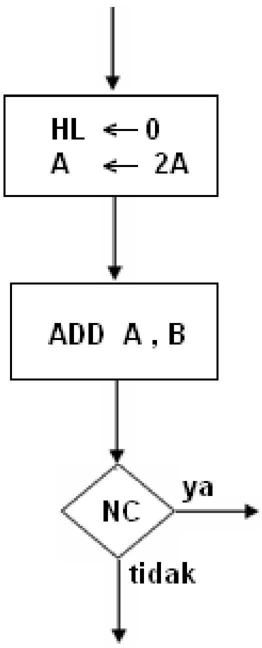
Suatu saat dibutuhkan sebuah algoritma untuk menggambarkan suatu proses yang berulang, misal pada suatu program dibutuhkan sebuah rutin yang

berfungsi sebagai penghitung (counter). Untuk itu flowchartnya dapat digambarkan sebagai berikut:



Gambar 5.5. Flowchart dengan proses diulang

Dalam membuat algoritma untuk pemrograman mikroprosesor dalam bahasa asembler biasanya notasi yang digunakan dalam flowchart menggunakan kode-kode, berikut contoh flowchart dengan notasi kode:



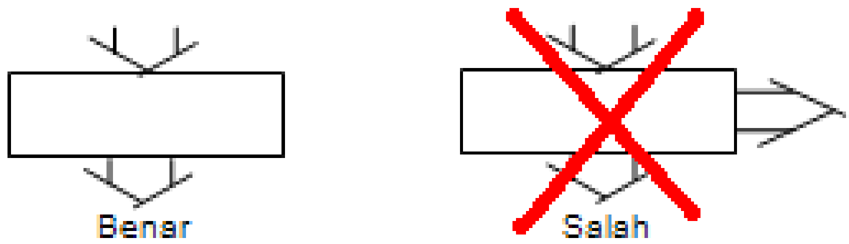
Gambar 5.6. Flowchart dengannnotasi kode program.

3. Struktogram

Dalam penggambaran algoritma melalui flowchart memberikan kejelasan pada kita bahwa setelah menganalisa masalah yang terjadi dengan tepat, maka dibuat sedikit atau banyak detail dari jalannya program yang telah direncanakan. Dengan ini kita dapat mengurangi kesukaran pemrograman, walaupun demikian pada saat kita membuat kesalahan logika pada suatu jalannya program, sering memberikan pelajaran yang baik untuk mendapatkan pemahaman dan jalan keluar lebih baik.

Sebuah perencanaan jalannya program menggambarkan urut-urutan proses atau jalannya fungsi sebuah program, perencanaan ini juga menunjukkan urutan yang mana atau syarat yang mana dari pelaksanaan kejadian sebelumnya, yang akan diulang. Selain penggambaran perencanaan jalannya program dituangkan dalam bentuk aliran program (Flow Chart), maka dapat pula dituangkan dalam bentuk penggambaran struktur program (Struktogram).

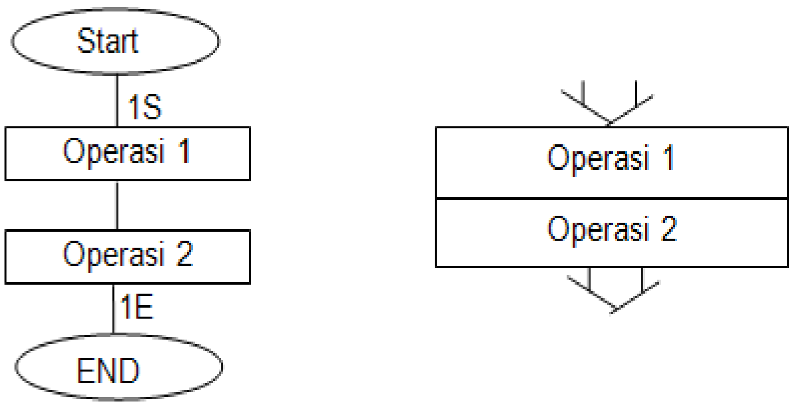
Tujuan dari penggambaran ini adalah untuk menghindari banyak kesalahan pada flow chart, bahwa masing-masing blok mempunyai sebuah masukan (S) dan sebuah keluaran (E). **Nassi Shneiderman** membangun sebuah bentuk khusus yang dikenal dengan "**Struktogram**" yang kemudian ditetapkan sebagai DIN 66261. Pada struktogram dibangun sistem penggambaran yaitu memiliki satu masukan dan satu keluaran.



Gambar 5.7 Penggambaran pada struktogram

Pada struktur program ada 3 bentuk konstitusi yang boleh digunakan sebagai bangun program.

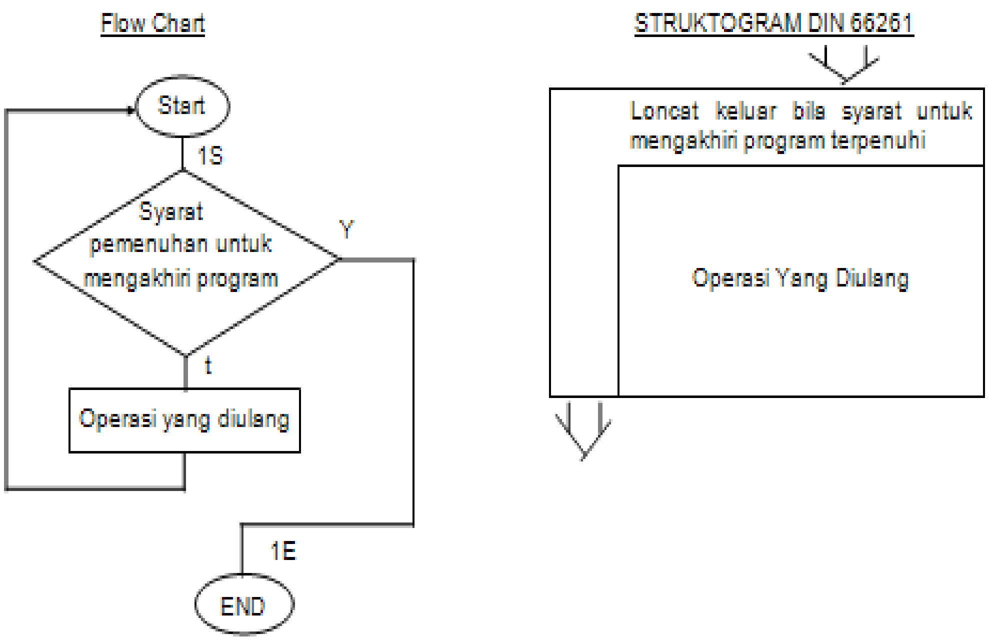
a. Struktur linier (berurutan/sequens)



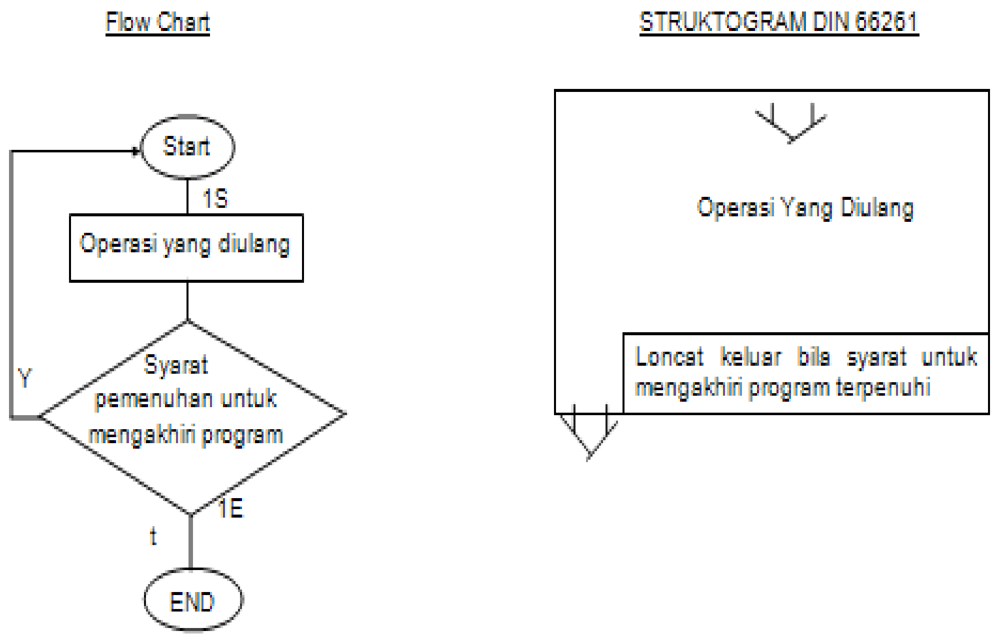
Gambar 5.8. Urutan perintah struktogram

Perintah-perintah yang ada dalam urutan perintah dalam pelaksanaannya berlangsung secara berurutan, dimana setelah pelaksanaan satu perintah maka isi PC (Penghitung Perintah) akan naik satu untuk melaksanakan perintah berikutnya.

b. Struktur Pengulangan (Loop)



Gambar 5.9. Pengulangan dengan syarat sebelum operasi pada struktogram



Gambar 5.10. Pengulangan dengan syarat sesudah operasi

Pengulangan program bagian yang dapat dijalankan berulang-ulang disebut sebagai tubuh dari pengulangan. Pada masing-masing pengulangan, minimal ada satu syarat loncat dan pada setiap pelaksanaan pengulangan, syarat loncat tersebut harus diuji.

Hasil pengujian akan mempengaruhi jalannya program bagian atau tubuh pengulangan atau jalannya perintah berikutnya.

Pengulangan pada umumnya terdiri dari beberapa bagian berikut ini :

- **Inisialisasi**
  - ➢ Pengisian register atau lokasi memori yang dipakai pada pengulangan.

  Contoh : alamat awal penyimpan data, nilai awal dari suatu penghitung.

  - ○ Pengosongan register atau flag.

  Bagian ini hanya dilaksanakan satu kali

- **Tubuh pengulangan**

  Hal ini tediri dari program yang harus berulang-ulang dilaksanakan.

  Contoh : Perhitungan, memeriksa kondisi masukan keluaran.

Rangka pengulangan ini dapat terdiri dari struktur pengulangan atau alternatip.

- **Aktualisasi dari Parameter Pengulangan**

Pada pemprosesan blok data, contoh untuk penunjuk alamat pada setiap pelaksanaan pengulangan diharuskan dinaikkan 1 atau diturunkan 1. Dengan demikian maka aktualisasi syarat untuk mengakhiri pengulangan untuk digunakan, sebagai beriku :

Penghitung, yaitu setiap pelaksanaan pengulangan isi register atau lokasi memori dinaikan atau diturunkan. Sampai pada pencapaian suatu nilai tertentu maka pengulangan akan berakhir.

- Demikian juga pada pengujian sebuah hasil (perhitungan bila sesuatu nilai tercapai, lebih besar atau lebih kecil maka pengulangan berakhir.
- Kemungkinan lain adalah pengujian bit kontrol dari isi register atau mencari, pengulangan dapat dipengaruhi melalui hasil dari jalannya program atau melalui pembacaan blok masukan keluaran.

- **Keputusan untuk mengakhiri pengulangan**

Untuk mengakhiri sebuah pengulangan sangat terkait dengan ketentuan atau kriteria keputusan yang telah ditentukan sebelumnya, sehingga setiap pelaksanaan operasi instruksi akan selalu melihat pada kondisi bit flag. Berdasarkan bit pada register flag inilah keputusan diambil, sebagai contoh hal ini terjadi pada saat pengujian pada akhir struktur pengulangan (Repeat Until).

Kekurangan pada instruksi ini, bahwa bagian yang diulang akan langsung berjalan pada saat masuk ke dalam pengulangan dan pengulangan berikutnya tergantung dari pemenuhan syarat.

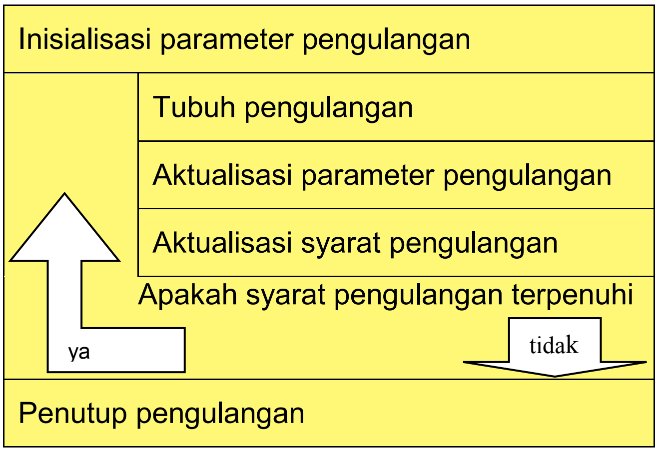
➢ **Pengujian pengulangan untuk memenuhi kriteria Repeat Until**

Algoritma secara narasi dan struktogram untuk mengakhiri pengulangan dapat dituliskan secara sekuensial sebagai berikut:

1) Inisialisasi parameter pengulangan
2) mengisi penghitung
3) mengisi alamat awal, menghapus register

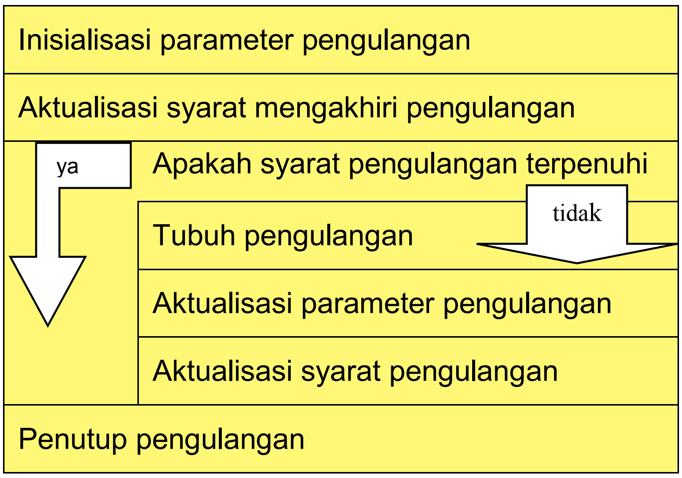
4) Tubuh pengulangan
5) Perhitungan
6) Pengurangan atau penambahan Penjunjuk alamat Aktualisasi parameter pegulangan
7) Pengurangan penghitung, membandingkan hasil dengan konstanta
8) Aktualisasi syarat pengulangan
9) Menguji bit flag yang dipakai sebagai syarat pengulangan
10) Penutupan pengulangan
11) Menyimpan hasil, mengambil kebali register

Struktogram dari algoritma dapat digambarkan sebagai berikut:



- **Pengujian Pengulangan pada awal struktur pengulangan (While DO)**

Tubuh pengulangan hanya akan dijalankan pada awal bila syarat pengulangan terpenuhi.



- **Penutup Pengulangan**

Di sini hasil yang didapat dari perhitungan pada saat pengulangan di simpan atau isi asli/awal dari register yang dipakai pada saat pengulangan di isi kembali pada register yang bersangkutan

Contoh 1:

Permasalahan :

Buatlah sebuah struktogram sebagai algoritma solusi permasalahan sebuah Mikroprosessor harus menambah 12 bilangan biner (panjang 8 bit) yang berada pada blok data, alamat awal blok data diberi simbol DATA. Hasil akhir penjumlahan diletakkan pada register HL. Alamat awal yang ditunjuk oleh Mikroprosessor : 0900 H, alamat DATA ; 0A00 H.

Pemecahan :

Berdasarkan permasalahan terdapat 12 bilangan biner, setiap bilangan 8 bit terletak pada blok memori dengan nama data beralamat di 0A00 H, alamat awal yang ditunjuk mikroprosesor 0900 H. Dengan permasalahan ini dapat ditentukan algoritma sebagai berikut:

- Penjumlahan harus terjadi pada tubuh pengulangan jumlah pengulangan harus dihitung pada register B.
- Karena penghitung, untuk memulai pengulangan tidak pernah diisi dengan 0, maka dipergunakan struktur repeat - until.
- Untuk blok data, diperlukan 1 penjumlah data, register IX
- Karena hasil dapat lebih besar dari 8 bit, maka Register HL dipakai sebagai tempat penghitung dan hasil.
- Operan tunggal /data 8 bit dapat diisi ke register E dan untuk kemudian ditambahkan dengan isi register HL untuk mendapatkan hasil sementara pada setiap pengulangan.

Parameter Masukan : alamat awal dari blok data (DATA)

Parameter Keluaran : HL berisikan hasil dari program

Register yang berubah : B, DE, HL, IX

Dalam gambar struktor algoritma tersebut dapat digambarkan sebagai berikut:

<table>
<tr><td colspan="2">Mengisi penghitung (reg B) dengan 0C H</td></tr>
<tr><td colspan="2">Mengisi reg IX sebagai penunjuk data dengan alamat awal 0A00 H</td></tr>
<tr><td colspan="2">Menghapus register HL dan E</td></tr>
<tr><td rowspan="4"></td><td>Mengisi reg E dengan data blok dan tambahkan dengan isi reg.HL</td></tr>
<tr><td>Menambah 1 penunjuk data reg IX, sebagai penunjuk data berikutnya</td></tr>
<tr><td>Penghitung pengulangan -1</td></tr>
<tr><td>Apakah penghitung pengulangan belum sama dengan 0</td></tr>
<tr><td colspan="2">Penutup pengulangan</td></tr>
</table>

Berdasarkan struktogram tersebut di atas dapat dirancang atau dibuat kode programnya dalam bahasa asembler, awal program dituliskan pada memori dengan alamat 0900H. Pada alamat ini register B digunakan sebagai peng-hitung, dengan memasukan data 0CH berarti panjang blok data yang akan diproses adalah sejumlah CH atau sama dengan 12 lokasi memori yang digunakan untuk menyimpan data.

Dilanjutkan pada pemberian atau pengisian data pada register IX yang dalam program ini digunakan untuk menunjuk alamat memori dimana data disimpan-kan, yaitu pada alamat 0A00 H. Oleh karena register yang digunakan untuk proses adalah register HL dan register E maka kedua register ini pada awal program sebelum digunakan sebagai register yang digunakan dalam proses dikosongkan terlebih dahulu.

Proses berikutnya menambahkan data yang ada dilokasi memori yang ditunjuk oleh register IX melalui register E dan register HL sampai register B=0 (nol), hal ini menyatakan bahwa data sudah habis dijumlahkan.

Program :

Tanda	Alamat (HEX)	Kode Operasi		Keterangan
		(HEX)	Mnemonik	
ADD 12	0900	06	LD B, 0CH	Jumlah data dalam blok data di isi ke register B
	0901	0C	-	
	0902	DD	LD IX,(DATA)	Alamat dari blok data yaitu Data dan DATA+1 ke register IX
	0903	2A	-	
	0904	00	-	
	0905	0A	-	
	0906	16	LD D, 00H	Menghapus register operan (D)
	0907	00	-	
	0908	62	LD H, D	Menghapus register operan (H)
	0909	6A	LD L, D	Menghapus register operan (L)
LOOP	090A	DD	LD E,(IX+0)	Mengisi reg.E dengan operan dari blok data yang alamatnya ditunjuk oleh IX
	090B	5E	-	
	090C	00	-	
	090D	19	ADD HL, DE	Menghasilkan hasil sementara
	090E	DD	INC IX	Alamat data berikutnya (IX+1)
	090F	23	-	
	0910	05	DEC B	Aktualisasi syarat pengulangan
	0911	C2	JPNZ, LOOP	menguji syarat pengulangan
	0912	0A	-	
	0913	09		
	0914	FF	HALT	Program selesai

Contoh 2 :

Permasalahan:

Mikroprosessor menampilkan urutan ke port keluaran. Alamat awal urutan karakter yang diberi simbol KATA adalah 0C00H. Alamat port keluaran yang diberi simbol P DATA adalah 00H. Data yang dipakai sebagai akhir dari data KATA adalah FFH. alamat awal Mikroprosessor : 0500H.

Pemecahan :

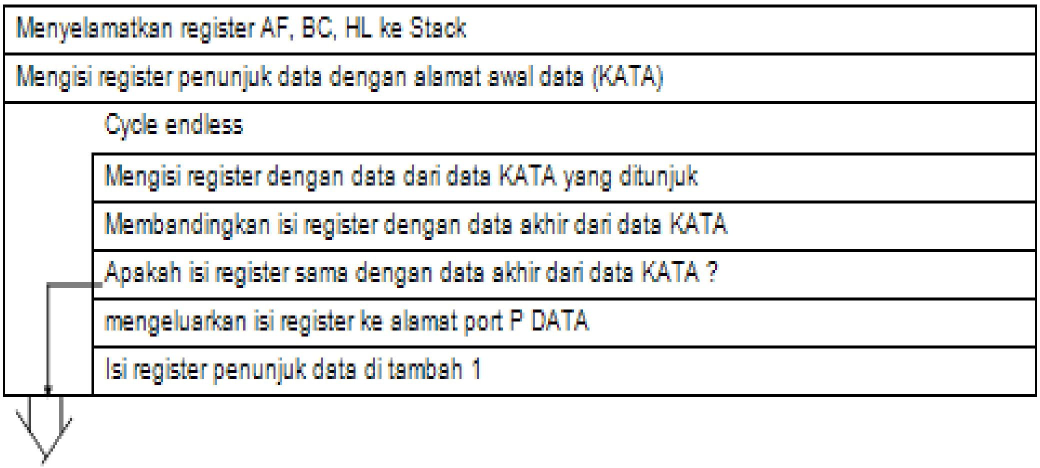
Parameter masukan : alamat awal dari data KATA

data sebagai tanda akhir urutan data adalah FFH.

Parameter keluaran : alamat port keluaran P DATA = 00H.

Register yang berubah : tidak ada.

STRUKTOGRAM

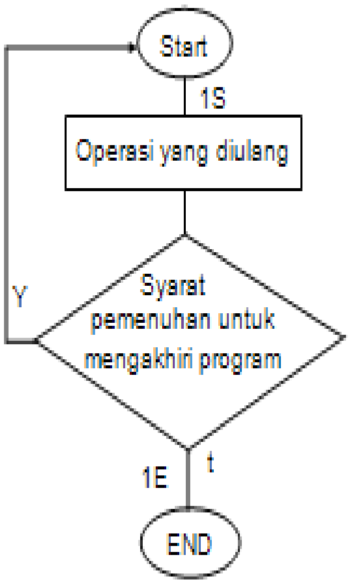


Program :

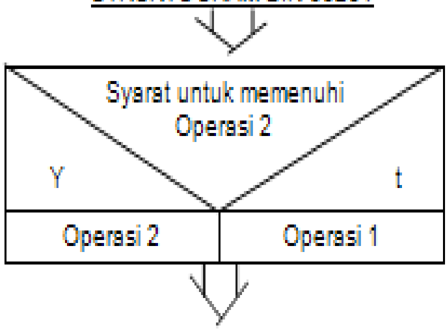
Tanda	Alamat (HEX)	Kode Operasi		Keterangan
		(HEX)	Mnemonik	
	0500	F5	PUSH AF	Menyimpan isi register AF,BC
	0501	C5	PUSH BC	dan HL ke stack
	0502	E5	PUSH HL	
	0503	21	LD HL,KATA	Mengisi register HL dengan alamat dari blok data
	0504	00	-	
	0505	0C	-	
	0506	0E	LD C, FFH	Mengisi reg.C dengan data akhir blok data
	0507	FF	-	
LOOP	0508	3E	LD A, (HL)	Mengisi reg A dengan isi memori yang ditunjuk oleh HL
	0509	B9	CPC	Membandingkan isi A dengan isi C
	050A	28	JPZ, END	Loncat ke label END, jika AKKU=FFH
	050B	14	-	
	050C	05		
	050D	D3	OUT (P DATA),A	Keluarkan isi AKKU ke port P DATA
	050E	00	-	
	050F	0C		
END	0510	23	INC HL	Isi reg HL ditambah 1
	0511	18	JP LOOP	Loncat tak bersyarat ke LOOP
	0512	08	-	
	0513	05	-	
END	0514	E1	POP HL	Mengambil kembali isi register
	0515	C1	POP BC	AF,BC dan H L dari stack
	0516	F1	POP AF	
	0517	FF	HALT	Program selesai

c. Struktur Keputusan (Percabangan, Alternatif)

Flow Chart

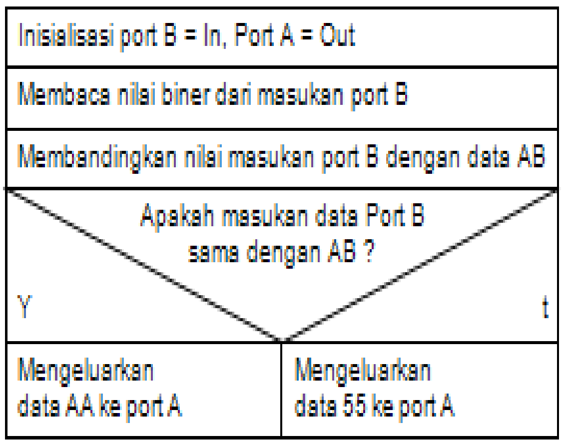


STRUKTOGRAM DIN 66261



Struktur ini terdiri dari sebuah blok pengontrol, yang telah ditentukan dan akan menjalankan alternatip bila syarat tertentu terpenuhi.

Contoh :



Parameter Masukan :

- Kata kendala 8255 untuk port A = Out dan port B = in adalah 82H
- Alamat port B = 01H

Parameter Masukan :

- Alamat port = 00H
- Alamat register kontrol = 03H

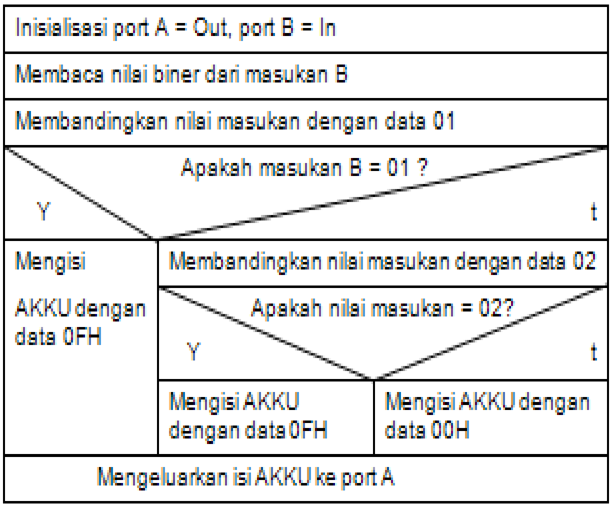
Alamat awal program 0600H

Program :

Tanda	Alamat (HEX)	Kode Operasi (HEX)	Kode Operasi Mnemonik	Keterangan
	0600	3E	LD A, 82H	Mengisi reg kontrol 8255 dengan data kendala 82H
	0601	82	-	
	0602	D3	OUT 03, A	
	0603	03	-	
	0604	DB	In A, 01H	Membaca nilai biner dari port B
	0605	01	-	
	0606	FE	CP, ABH	Membandingkan nilai yang dibaca dari port B dengan konstanta ABH
	0607	AB	-	
	0608	CA	JPZ,0612	Loncat ke alamat 0612 untuk mengeluarkan data AA ke port A, jika nilai yang dibaca = ABH
	0609	12	-	
	060A	06	-	
	060B	3E	LD A, 55H	Mengeluarkan data 55H ke port A
	060C	55	-	
	060D	D3	OUT 00H, A	
	060E	00	-	
	060F	C3	JP 0616	Loncat tak bersyarat untuk mengakhiri program
	0610	16	-	
	0611	06	-	
	0612	3E	LD A, AAH	Mengeluarkan data AAH ke port A
	0613	AA	-	
	0614	D3	OUT 00, A	
	0615	00	-	
END	0616	FF	HALT	Program selesai

Contoh 2 : .

STRUKTOGRAM



<u>Parameter Masukan :</u>

- Kata kendala 8255 untuk port A = Out dan port B = Input
- Alamat port B = 01H

<u>Parameter Masukan :</u>

- Alamat port = 00H
- Alamat register kontrol = 03H

Alamat awal program 1800H

Program :

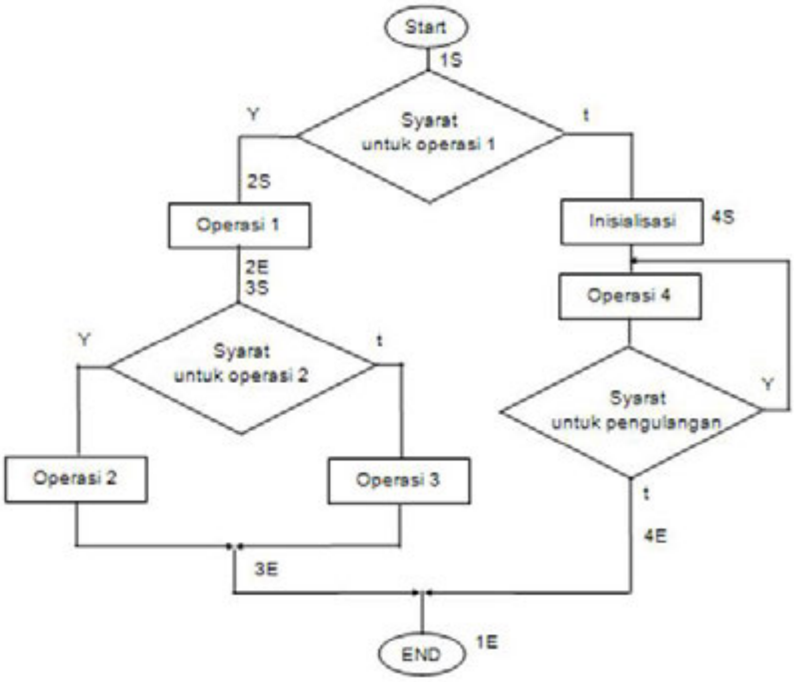
Tanda	Alamat (HEX)	Kode Operasi		Keterangan
		(HEX)	Mnemonik	
	1800	3E	LD A, 82H	Mengisi register kontrol 8255 dengan kata kendala 82H
	1801	82	-	
	1802	D3	OUT 03H, A	
	1803	03		
	1804	DB	IN A, 01H	Membaca nilai biner dari port B
	1805	01	-	
	1806	FE	CP 01H	Membandingkan nilai yang dibaca dari port B dengan konstanta 01H
	1807	01	-	
	1808	CA	JPZ, 181AH	Loncat ke alamat 181AH untuk mengisi akku dengan data 0F, jika nilai yang dibaca = 01H
	1809	19	-	
	180A	18	-	
	180B	FE	CP 02H	Membandingkan nilai yang dibaca dari port B dengan konsta 02H
	180C	02	-	
	180D	CA	JPZ, 1815H	Loncat ke alamat 1815H untuk mengisi akku dengan data F0H, jika nilai yang dibaca = 02H
	180E	15	-	
	180F	18	-	
DOO	1810	3E	LD A, 00H	Mengisi akku dengan data 00H
	1811	00	-	
	1812	C3	JP 181CH	Loncat tak bersyarat untuk mengeluarkan isi akku ke port A
	1813	1B	-	
	1814	18	-	
DFO	1815	3E	LD A, F0H	Mengisi akku dengan data F0H
	1816	F0	-	
	1817	C3	JP 181CH	Loncat tak bersyarat untuk mengeluarkan isi akku ke port A
	1818	1B	-	
	1819	18	-	
	181A	3E	LD A, 0FH	Mengisi akku dengan data 0FH
DOF	181B	0F	-	
	181C	D3	OUT 00, A	Mengeluarkan isi akku ke port A
	181D	00	-	
END	181E	FF	HALT	Program selesai

d. Struktur Struktogram Kombinasi.

Dari ketiga macam struktur dalam struktogram, yaitu struktur linier sekuensial, struktur pengulangan (Loop), struktur keputusan atau percabangan dengan

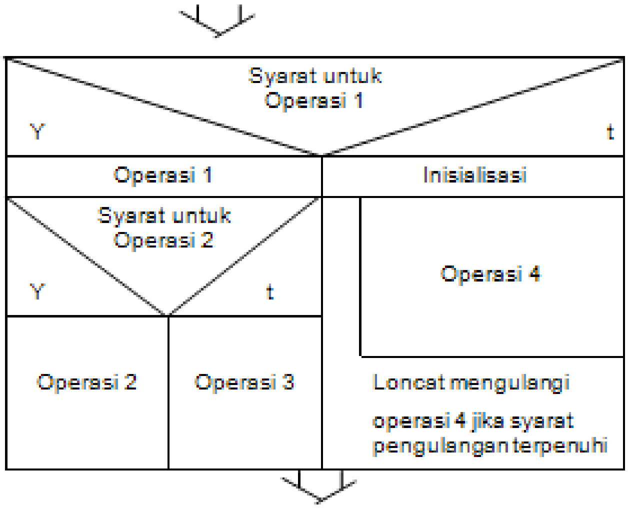
alternatip dapat digunakan secara gabungan atau kombinasi. Hal ini dilakukan manakala ditemui permasalahan yang harus diselesakan dengan sebuah program yang agak komplek, oleh karena itu struktogram dapat disusun dan dikembangkan dengan kombinasi dari ketiga bentuk diatas.

Berikut merupakan flowchart dari permasalahan yang mengharuskan adanya 4 macam operasi, yaitu operasi 1, operasi 2, operasi 3 dan operasi 4. Sedang alternatip pilihan untuk setiap operasi dilakukan secara berjenjang, alternatip pertama jika tidak memenuhi syarat alternatipnya menuju operasi 4 dan jika syarat alternatip terpenuhi menuju operasi 1. Pada operasi 4 terdapat proses pengulangan jika syarat terpenuhi dan jika tidak terpenuhi maka program berhenti (END), Pada operasi 1 dilanjutkan dengan sebuah alternatip dengan syarat terpenuhi operasi dilanjutkan ke operasi 2, dan jika tidak terpenuhi dilanjutkan dengan operasi 3 kemudian program berhenti (END).



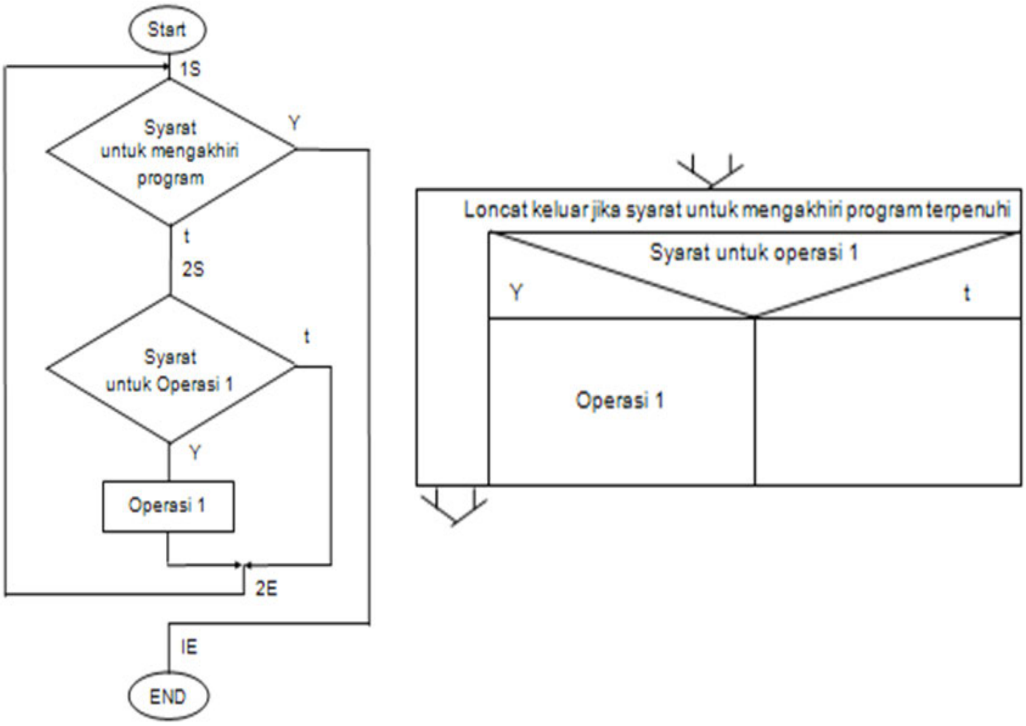
Gambar 5.11. Flowchart untuk permasalahan komplek.

Bentuk algoritma dalam struktogram dari permasalahan pada flowchart di atas dapat digambar sebagai berikut:



Gambar 5.12. Struktogram untuk permasalahan komplek.

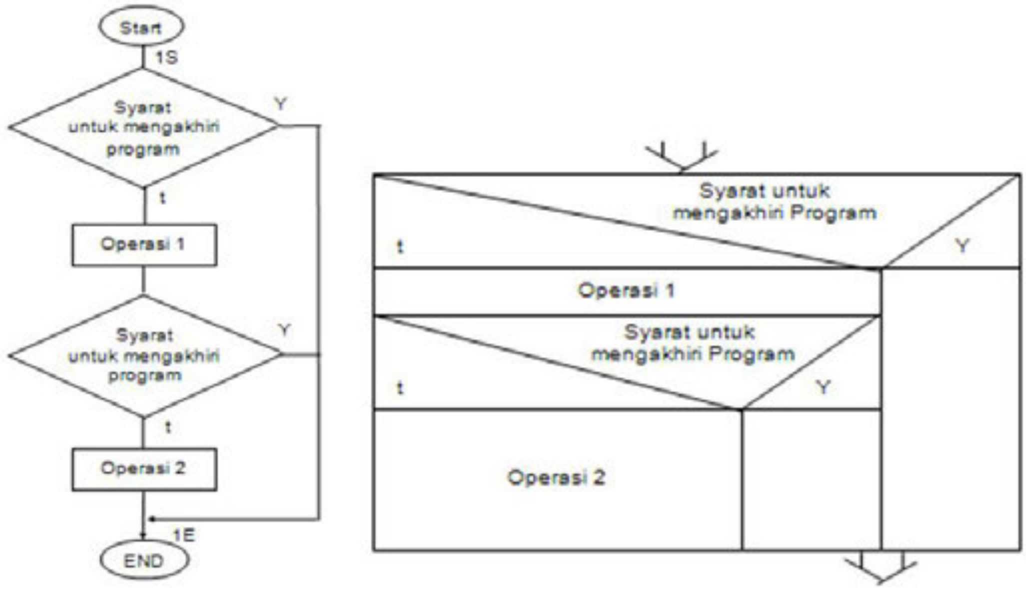
Contoh berikut merupakan flowchart dan struktogram dari solusi permasalahan yang mengharuskan memilih operasi 1 atau menghentikan program.



a. Flowchart b. Struktogram

Gambar 5.13. Algoritma pilihan berhenti atau operasi

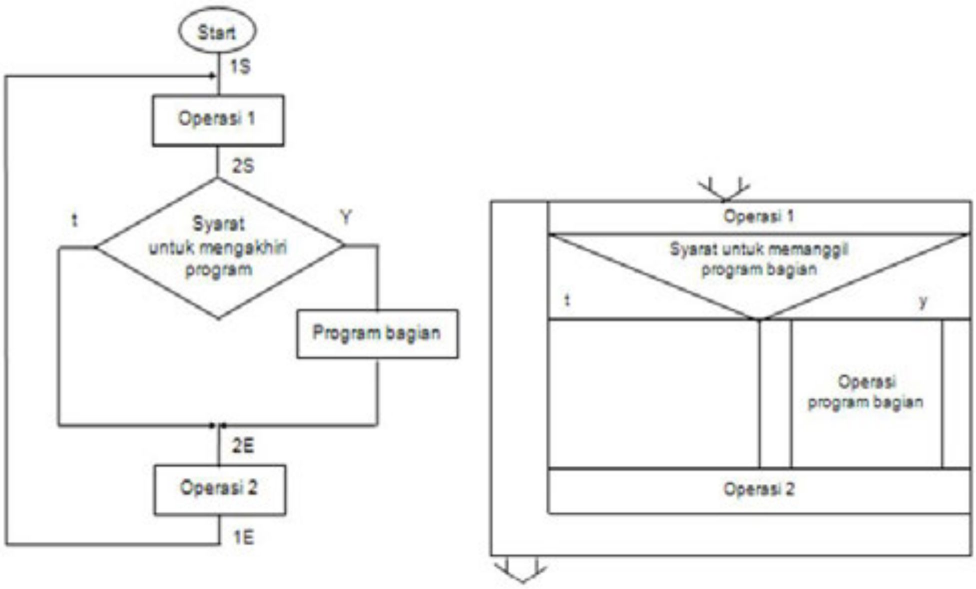
Berikut sebuah contoh untuk menghentikan 2(dua) buah operasi, yaitu operasi pertama jika syarat dipenuhi dan operasi kedua berhenti jika syarat dipenuhi.



a. Flowchart b. Struktogram

Gambar 5.14. Algoritma pilihan berhenti dari dua operasi

Berikut merupakan contoh solusi untuk pemanggilan program bagian.



a. Flowchart b. Struktogram

Gambar 5.15. Algoritma pemanggilan program bagian

2.1.1 Soal Latihan:

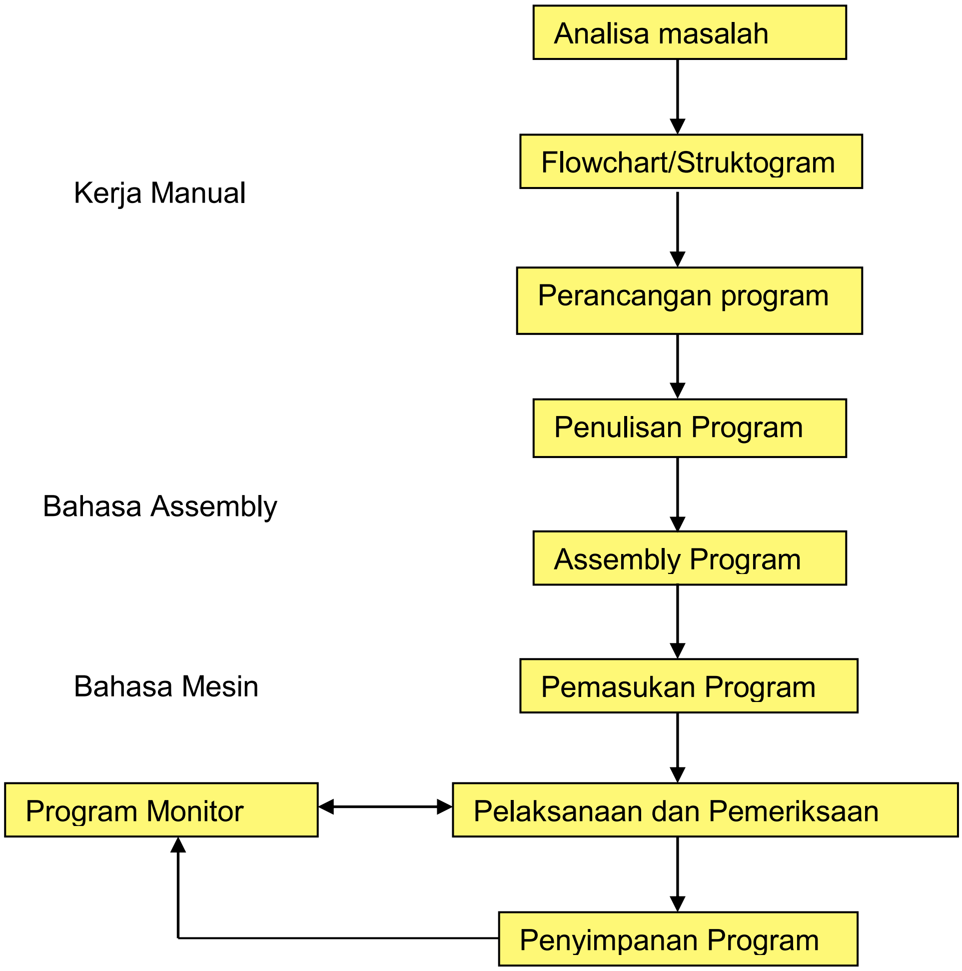
1. Buatlah flow chart dan struktogram untuk menyelesaian permasalahan mengelompokan nilai ganjil dan genap dari bilangan mulai dari 1 sampai 1000!
2. Buatlah flow chart dan struktogram untuk menyelesaian permasalahan mencari bilangan prima dari bilangan 1 sampai 100!
3. Buatlah flow chart dan struktogram dari sebuah mesin penjual minuman kopi dan teh otomatis!
4. Buatlah flow chart dan struktogram dari sistem pengatur lampu lalu lintas pada sebuah perempatan jalan, dengan ketentuan jalan merupakan jalan dua arah!
5. Buatlah flow chart dan struktogram dari sebuah sistem Lift 4 tingkat, sehingga seorang pengguna dapat memilih tingkat berapa yang dituju dan saat itu berada pada tingkat berapa!

2.2 MERANCANG PROGRAM MIKROKOMPUTER

Suatu program mikrokomputer merupakan susunan sejumlah instruksi atau perintah. *CPU(Central Processing Unit)* komputer tersebut akan melaksanakan langkah-langkah logik untuk mencapai hasil yang diinginkan. Sebelum suatu program dilaksanakan oleh CPU, program tersebut harus disimpan di memori dalam bentuk biner. Program jenis ini disebut program dalam bahasa mesin *(machine language program).* Hanya jenis bahasa inilah yang dapat dimengerti oleh sebuah mikrokomputer. Program dalam bahasa mesin ini biasanya dinyatakan dalam digit hexadesimal. Misalnya, instruksi 8 bit 1010 1111 B (B menyatakan biner) dalam CPU Z80 dapat diganti dengan AF H (H menyatakan hexadesimal). Bagi pemakai, menginterpretasikan suatu program dalam bahasa mesin sangatlah sulit dan membutuhkan banyak waktu. Para pembuat mikroprosesor telah membagi instruksi-instruksi tersebut menjadi beberapa kategori menurut fungsinya. Instruksi-instruksi CPU dan register-register biasanya dinyatakan dalam simbol-simbol yang disebut *"mnemonics".* Misalnya, instruksi CPU Z80 70H dapat dinyatakan dalam kode *mnemonic* LD A,L (artinya memasukkan data dari register L ke register A). Suatu program yang ditulis dalam kode *mnemonic* disebut program dalam bahasa *ussemblv.* Sebelum suatu program dalam bahasa *assembly* dilaksanakan oleh CPU, program tecsebut harus diterjemahkan dalam bahasa mesin oleh program khusus yang disebut *"Assembler".*

Biasanya suatu program ditulis dalam bahasa *assembly.* Keunggulan utama dari program dalam bahasa *assembly* terhadap program dalam bahasa mesin ialah bahwa program dalam bahasa *assembly* jauh lebih cepat membuatnya, *mnemonic mnemonicnya* membuat para pemakai lebih mudah mengingat instruksi set-nya, dan biasanya *assembler* telah mempunyai paket *"self-diagnostic"* untuk memeriksa program yang dibuat apabila ada kesalahan. Kekurangan utama program dalam bahasa *assembly* ialah bahwa dia membutuhkan sebuah *assembler* (penerjemah ke bahasa mesin) dan alat/sistem pengembangan mikrokomputer, yang pada umumnya sangat mahal. Dengan mikrokomputer uPro-1, para pemakai harus menerjemahkan bahasa *assembly* ke bahasa mesin dengan melihat tabel sebelum menuliskan program tersebut pada uPro-1 dan menjalankannya.

Dalam merancang suatu program biasanya kita melalui tahap-tahap seperti terlihat pada diagram alir perancangan yang dapat dilihat pada gambar dibawah ini :



Gambar 5.16. Tahapan dalam desain program mikroprosesor

1. Analisa Masalah

Program untuk suatu masalah sederhana dapat dibuat dengan mudah melalui *flowchart* yang tersusun baik. Dapat juga dibuat dengan merevisi (mengganti beberapa bagian) program yang telah ada atau dengan cara menggabungkan *routine-routine* sederhana. Merancang program-program yang lebih rumit, misalnya program-program sistem kontrol atau program dengan fungsi khusus, biasanya didahului dengan membuat analisa terperinci dari masa(ah yang bersangkutan.

Analisa masalah dan pemecahannya membutuhkan pengertian yang sungguh-sungguh akan beberapa hal seperti dibawah ini: .

1) Karakteristik dan tuntutan masalah
2) Kondisi-kondisi yang telah diketahui
3) Format informasi input dan bagaimana format itu dikonversikan
4) Format data output dan bagaimana format itu dikonversikan
5) Jenis data dan ketelitiannya
6) Waktu pelaksanaan program yang dibutuhkan
7) Instruksi-instruksi CPU dan sifat-sifatnya
8) Besarnya memori
9) Kemungkinan dapat/tidaknya masalah tersebut diselesaikan
10) Metoda pemecahan masalah
11) Evaluasi program
12) Bagaimana/dimana hasil pembuatan program akan disimpan

2. Merancang Program

Ada beberapa jenis program. Program persamaan matematik, konversi sinyal *input* dan *output,* program untuk mengkodekan dan men-dekode data, program untuk menjalankan peripheral adalah contoh program-program yang sederhana. Program-program untuk *assembler,* kontrol sistem atau aplikasi pada peralatan-peralatan khusus adalah contoh rumit. Dalam mecancang suatu program kita biasanya memperhatikan hal-hal dibawah ini:

1. Membaca sinyal *input* atau data
2. Menghasilkan atau mengkonversi sinyal *output dan data*
3. Perhitungan dan analisa logika di dalam program utama
4. Hubungan antara program utama dan *subroutine*
5. Penggunaan dari register-register yang ada di dalam CPU
6. Alokasi penggunaan memori pada program utama

7. Alokasi penggunaan memori pada *subroutine*

8. Alokasi memori untuk tabel data dan metoda index addressing

9. Program inisialisasi dan konstanta-konstanta

10. Definisi dari variabel-variabel di dalam program

11. Pertimbangan dari urutan waktu dan kecepatan pelaksanaan program

12. Keterbatasan kapasitas memori

13. Panjang dan kepresisian data

14. Dokumen dan bahan rujukan yang ada (tersedia)

15. Hal-hat khusus lain

3. Penulisan Program

Dalam buku ini, program-program umumnya ditulis dalam bahasa *assembly* dan hanya format program dalam bahasa *assembly* yang dijelaskan disini.

Suatu *statement* dalam program terdiri dari empat bagian: label, opcode, *operand* dan keterangan. Lihat contoh dibawah ini:

Label *Opcode* *Operand* Keterangan

```
DTB4    LD      B,16
DB3     SRL     H
        RR      L
        RR      D
        RR      E       ;rotasikan HL DE ke kanan
        LD      A,H
        CALL    DB1
        LD      H,A     ;kembalikan isi H seperti sebelum CALL DB1
```

```
        LD      A,L
        CALL    DB4
        LD      L,A     ;kembalikan isi L seperti sebelum CALL DB4
        DJNZ    DB3
        RET
```

Routine pembetulan biner

```
DB4     BIT     7,A
        JR      Z,DB1   ;jika A BIT 7 = I,
        SUB     30H     ;kurangkan dengan 30H
DB1     BIT     3,A
        JR      Z,DB2   ;jika A BIT 3 = I,
        SUB     03H     ;kurangkan dengan 03H
DB2     RET
```

Suatu *statement* program tanpa keterangan kadang-kadang sulit di-mengerti. Keterangan tersebut lebih dirasakan artinya pada program-program yang rumit. Suatu *statement* dengan label dan keterangan lebih mudah dipanggil atau diperiksa.

4. Program *Assernbly*

Cara yang paling efektif dalam menerjemahkan program asal ke bahasa mesinnya adalah dengan mempergunakan *assembler* yang telah tersedia dalam sistem mikrokomputer tersebut. Tetapi, seorang pemula atau *programmer* yang belum mengenal sistem pengembangan mikrokomputer tersebut dapat pula menerjemahkan program manual, prosedurnya adalah sebagai berikut :

1) Terjemahkan setiap instruksi *(opcode* dan *operand)* ke dalam kode mesin dengan melihat tabel konversi.

2) Setelah menentukan alamat awal program, tentukan alamat yang sesuai untuk byte pertama masing-masing instruksi. Jumlah *byte* yang dibutuhkan secara tepat harus disediakan termasuk juga *byte-byte* yang akan diisi kemudian seperti misalnya dalam instruksi JR, DJNZ, serta alamat tujuan dalam instruksi JP, CALL, dan lain- lain.
3) Menghitung percabangan relatif dan mengisikannya ke dalam program yang sudah *di-assembly.* Rumus sederhana dibawah ini dapat digunakan untuk menghitung percabangan relatif (pergeseran)
4) Pergeseran = alamat tujuan - alamat instruksi berikutnya

Jika hasil perhitungan adalah positif, maka hasil perhitungan tersebut adalah nilai yang kita isikan. Jika hasil perhitungan adalah negatif, maka tambahkan 100H dengan hasil tersebut (akan diperoleh nilai komponen keduannya) dan hasil akhirnya merupakan hasil yang dapat kita isikan. Contohnya, dalam program yang tertulis diatas instruksi DJNZ DB3 pada alamat 0014H mula-mula diterjemahkan menjadi 10xx dan kemudian nilai xx dihitung sebagai berikut

xx = 0002H (alamat tujuan) - 0016H (alamat instruksi berikutnya)

= -14H (nilai negatif)

xx = 100H + (-14H) = 100H - 14H = 0ECH

Karena itu instruksi DJNZ DB3 harus diterjemahkan menjadi 10EC. Demikiaa pula instruksi JR Z,DBI pada alamat 0019H mula-mula diterjemahkan menjadi 28xx kemudian kita menghitung nilai xx

xx = 001DH (alamat tujuan) - 001BH (alamat instruksi berikutnya)

= 2H

Jadi terjemahan instruksi JR Z,DB1 adalah 2802

Dibawah ini adalah contoh program dengan bahasa mesinnya

```
Address  B.Mesin   Label  Opcode  Operand   Keterangan;
;**Routine Konversi 4 Digit BCD ke biner
        ;input: data BCD di HL
        ;output: data biner di DE
;register yang berubah : AF BC DE HL
0008     CB1B             RR      E         ;rotasikan HL DE ke kanan
000A     7C               LD      A,H
000B     CD1D00           CALL    DB1
000E     67               LD      H,A       ;kembalikan isi H seperti sebelum
                                            ;CALL DB1
000F     7D               LD      A,L
0010     CD1700           CALL    DB4
0013     6F               LD      L,A       ;kembalikan isi L seperti sebelum
                                            ; CALL DB4
0014     10EC             DJNZ    DB3
0016     C9               RET
;Routine Pembetulan Biner
0017     CB7F      DB4    BIT     7,A
0019     2802             JR      Z,DB1     ;jika A BIT 7 = 1, kurangkan
                                            ;dari 30H
001B     D630             SUB     30H
001D     CB5F      DB1    BIT     3,A
001 F    2802             JR      Z,DB2     ;jika A BIT 3 = 1, kurangkan
                                            ;dari 03H
0021     D603             SUB     3
0023     C9        DB2    RET
0000     0620      DTB4   LD      B,1 6
0002     CB3C      DB3    SRL     H
```

0004	CB1D	RR	L
0006	CB1A	RR	D

5. Pengisian Program

a. Pengisian Program

Program monitor dapat digunakan untuk membantu para pemakai dalam mengisikan program ke alamat memori yang tersedia dalam uPro-I. Input program dapat diambil dari *keyboard* atau dibaca dari *tape recorder.* Setelah program diisikan ke dalam RAM uPro-I, proses pemeriksaan kesalahan diperlukan untuk menghilangkan kesalahan-kesalahan yang mungkin terjadi. Instruksi-instruksi atau data yang tidak berpengaruh dapat diganti dengan instruksi NOP (00). instruksi data yang tertinggal dapat diselipkan pada alamat yang diinginkan dengan menggunakan penyelipan data ([NS) atau dengan mengisikan kembali program tersebut. Bila merevisi suatu program, kita perlu memeriksa apakah instruksi-instruksi loncat (Jump), yaitu JP, JR, DJNZ. CALL, dan sebagainya terpengaruh oleh perubahan alamat-alamat pada memori. Jika hal ini terjadi, kita perlu memperbaikinya dengan segera.

b. Menjalankan Program dan Memeriksa

Sebelum menjalankan suatu program, kita perlu mengeset parameter--parameter inisialisasi dan meletakkan penghitung program (Program *Counter)* pada alamat awal program. Dengan menekan `GO', pelaksanaan program akan dimulai. Setelah program selesai dilaksanakan, periksalah hasilnya. Jika ada kesalahan, program harus diperiksa langkah demi langkah dengan bantuan z program monitor. Setelah program selesai direvisi, jalankan sekali lagi dan periksa hasilnya pula.

2.2.2 Latihan:

1) Tulislah sebuah program dalam bahasa assembly untuk men-set register sebagai berikut A=0, B=1, C=2, D=3, E=4, H=5 dan L =6, gunakan instruksi LD 8 bit untuk setiap kali pengisian ke dalam register dan pada akhir program gunakan instruksi RST 38H.
2) Tulislah sebuah program dalam bahasa assembly untuk men-set register sebagai berikut A=12, B=1A, C=D2, D=23, E=34, H=75 dan L =F6, gunakan instruksi LD 16 bit untuk setiap kali pengisian ke dalam register dan pada akhir program gunakan instruksi RST 38H.
3) Tulislah program untuk menghapus isi memori mulai memori beralamat 1850H sampai memori beralamat 1870H, jika digunakan instruksi LD 8 bit maka akan tertulis sebanyak 1870H-1850H = 20 H. Oleh karena itu tulis program menggunakan metode loop sehingga program lebih pendek!
4) Tulislah program untuk menjumlahkan isi register D dengan isi register E, kemudian hasilnya letakan pada pasangan register HL!
5) Tulislah program untuk menjumlahkan isi memori beralamat di 1900 dengan isi register pasangan BC dan hasilnya simpan di memori pada alamat 1A00!
6) Tulislah program untuk menjumlah data 32 bit yang berada pada memori 1900H sampai 1903H dengan isi memori 1A00H sampai 1A03H dan hasilnya letakan pada memori beralamat 1905H!
7) Tulislah program untuk menggeser ke kiri 4 kali pada data 8 bit yang disimpan pada alamat 1900H sampai 1920H, gunakan 2(dua) tingkatan loop yaitu loop dalam dan loop luar!
8) Tulislah program untuk mengalikan data 8 bit pada register E dengan isi register A dan letakan hasilnya pada register pasangan HL!

PEMROGRAMAN MIKROPROSESOR

DESKRIPSI MATERI PEMBELAJARAN

Pemrograman mikroprosesor pada hakekatnya adalah kelanjutan dari pembuatan rancangan program yang diperuntukan pada fungsi tertentu, fungsi tersebut digunakan untuk acuan dasar pembuatan berbagai program aplikasi yang diterapkan pada sebuah *hardware* mikroprosesor. Sebuah program aplikasi dikembangkan dengan cara menyusun berbagai instruksi-instruksi, dan melalui instruksi inilah komponen-komponen sebuah mikroprosesor dikendalikan. Seperti dijelaskan pada bab terdahulu bahwa program aplikasi pada mikroprosesor dapat difungsikan untuk memenuhi keperluan mulai dari kendali peralatan rumah tangga, sistem keamanan mobil dan rumah sampai pada sistem kendali di bidang militer dan bidang industri. Untuk membangun program-program aplikasi tersebut diperlukan sebuah bahasa pemrograman yang disebut dengan bahasa assembly, dan syarat utama dalam membuat program harus mengikuti tata aturan dan konsep yang terstruktur berupa algoritma pemrograman dan dituangkan dalam bentuk program aplikasi.

KOMPETENSI INTI (KI-3)	KOMPETENSI INTI (KI-4)
Kompetensi Dasar (KD): 2. Menerapkan pemrograman input output analog digital ***Indikator:*** 1.3. Memahami pemrograman input output analog. 1.4. Memahami pemrograman input output digital	**Kompetensi Dasar (KD):** 2. Membuat pemrograman mikro-prosesor input-output analog digital ***Indikator:*** *1.3.* Membuat program input-output analog dengan menggunakan perangkat lunak dan interpre-tasi data hasil pemrograman. *1.4.* Membuat program input-output digital dengan menggunakan perangkat lunak dan interpre-tasi data hasil pemrograman.

KATA KUNCI PENTING

- input-output, analog, digital
- pemrograman

BAB III. PEMROGRAMAN MIKROPROSESOR

3.1. PEMROGRAMAN MIKROPROSESOR

Sebagai pijakan awal dalam memprogram sebuah mikroprosesor adalah dengan memahami bahasa pemrograman, melalui bahasa pemrograman inilah programmer dapat menyusun instruksi-instruksi secara sekuensial guna memerintah mikroprosesor untuk melakukan suatu proses.

Melalui bahasa pemrograman seorang programmer dapat menentukan tahapan logika dan perhitungan pada mikroprosesor, oleh karena itu bahasa pemrograman dilengkapi dengan sekumpulan atau himpunan tata aturan sintaks dan semantik instruksi untuk menyusun suatu program komputer.

Terdapat 2(dua) bagian besar dalam sebuah program yaitu data dan program itu sendiri, yang kedua-duanya disimpan dalam memori baik memori RAM maupun ROM. Dengan program dapat diolah ataupun dikelola berbagai data tersebut, bagaimana data disimpan, dibaca, dibedakan jenisnya dan dengan cara atau langkah apa yang harus dilakukan di dalam mikroprosesor.

Seperti dijelaskan pada bab terdahulu bahwa mikroprosesor tersusun dari berbagai komponen, dan komponen-komponen inilah sebagai pembentuk mesin pengolah data yang disebut sebagai mikroprosesor. Oleh karena itu bahasa pemrograman mikroprosesor dibedakan sebagai berikut:

1. Bahasa Mesin, yaitu pemrograman dilakukan dengan penyusunan instruksi atau perintah dalam kode biner, yaitu dengan memberikan logika 0 dan atau 1.
2. Bahasa Tingkat Rendah atau dikenal dengan istilah bahasa assembler, yaitu pemrograman dilakukan dengan penyusunan instruksi atau perintah dalam kode yang bisa dikenali oleh manusia melalui *mnemonic.*
3. Bahasa Tingkat Tinggi, yaitu bahasa pemrograman komputer yang instruksinya lebih mendekati kata yang digunakan dalam bahasa manusia sehari-hari, misal do, print, if, for, repeat,until, AND, OR.

Sedangkan yang dimaksud dengan program adalah serangkaian instruksi yang disusun sehingga dapat melakukan sebuah fungsi dalam mikroprosesor, dan setiap mikroprosesor agar dapat melakukan suatu proses membutuhkan

keberadaan program. Sebuah program memiliki suatu pola atau bentuk model tertentu sesuai dengan bahsa yang digunakan, dengan format kode tersebut isi program dapat dipahami oleh programer yang sering dberi istilah kode sumber (*source code*).

Pemrograman mikroprosesor meliputi kegiatan menganalisis permasalahan melakukan pembuatan algoritma yang selanjutnya diwujudkan dalam bentuk program, selanjutnya program dikompilasi untuk bisa dikenali oleh mikroprosesor sebagai mesin. Dengan demikian program siap untuk dijalankan, biasa-nya sebuah program yang telah dibuat tidak berjalan dengan semestinya oleh karena itu perlu dilakukan uji program (*debug*).

Sebuah program disimpan terlebih dahulu dalam memori utama (RAM) sebelum dijalankan, kemudian mikroprosesor akan mengeksekusi program tersebut. Eksekusi dilakukan secara instruksi demi instruksi sampai seluru instruksi dalam program dijalankan atau sampai program tersebut dihentikan. Sebuah program sedang berjalan dalam mikroprosesor dapat berhenti, dan penghentian jalannya sebuah program dapat dilakukan berdasarkan permintaan, interupsi, adanya kesalahan dalam pemberian instruksi program, adanya permasalahan pada perangkat keras.

Program membutuhkan algoritma, karena pembuatan program membutuhkan tahapan solusi sekuensial selalu guna menyelesaikan masalah. Dengan berbasis pada logika pikir maka dapat dibuat suatu penyelesaian (solusi) terhadap suatu permasalahan. Sehingga algoritma mutlak dibuat dalam membuat suatu program, karena terdapat tahapan atau urutan langkah-langkah yang didalamnya berisi solusi logis penyelesaian masalah dan tersusun secara sistematis sehingga dapat mencapai tujuan yang diharapkan.

Dari uraian di atas dapat disimpulkan bahwa pemrograman merupakan suatu proses iteratif penulisan dan penyuntingan kode sumber sehingga membentuk sebuah program, setelah didapatkannya algoritma penyelesaian masalah. Program hakekatnya penyusunan kode sumber dilanjutkan proses pengujian, analisis, pembetulan kesalahan, optimalisasi algoritma, normalisasi kode.

Telah dijelaskan pada bab sebelumnya bahwa bagaimana proses dan dari mana sumber yang dapat dimanfaatkan dalam menyusun rencana dan program pada

mikroprosesor menjadi pertanyaan yang sangat penting, hanya dengan logika akal pikiran manusia dapat mewujudkan sebuah rancangan program yang terstruktur, jelas, logis dan mempunyai target dan tujuan yang jelas. Pada kenyataannya hampir tidak ada batasan manusia dalam memanfaatkan logika dan akal pikirannya untuk mengkreasi sesuatu yang bermanfaat bagi dirinya maupun orang lain. Seperti diketahui logika dan akal pikiran manusia mampu bekerja selama 24 jam terus menerus, dari kenyataan tentang kerja logika dan akal pikiran manusia ini maka sudah seharusnya kita bersyukur dan mau mengakui keagungan-NYA

Untuk mncapai apa yang diinginkan atau untuk memenuhi kebutuhan yang diharapkan pada sistem mikroprosesor, maka untuk dapat digunakan sebagai alat bantu dalam menyelesaikan masalah harus diprogram dengan suatu alur penyelesaian masalah yang dirancang sebelumnya. Ditinjau dari disiplin ilmu maka pemrograman dikenal dengan istilah rekayasa perangkat lunak (*Software engineering*).

3.2. PEMROGRAMAN BERBASIS MASALAH

Dalam pembahasan berikut dijelaskan pemrograman mikroprosesor berbasis masalah, penyelesaian masalah diawali dengan penyusunan struktogram kemudian dikodekan menjadi bentuk program asembler, dan pada akhirnya didapat suatu hasil akhir paskah dijalankannya program tersebut.

1. Program Transfer Data

a. Masalah 1 :

Mengisi register-register A = 00H, B = 01H, C = 02H, D=03H, E=04H. Menyalin isi register B dan C ke register H dan L.

Program dimulai pada alamat 1800H.

Program ditutup dengan HALT.

Penyelesaian :

- **Struktogram.**

Mengisi register A = 00H
Mengisi register B = 01H
Mengisi register C = 02H
Mengisi register D = 03H
Mengisi register E = 04H
Menyalin isi register B ke register H
Menyalin isi register C ke register L
Mengisi SP = 1F9FH
Mengakhiri program

- **Program**

Label	Alamat (HEX)	Kode program Op-Code			Mnemonik	Keterangan
	1800	3E	00		LD A, 00H	mengisi register A dengan data 00
	1802	06	01		LD B, 01H	mengisi register B dengan data 01
	1804	0E	02		LD C, 02H	mengisi register C dengan data 02
	1806	16	03		LD D, 03H	mengisi register D dengan data 03
	1808	1E	04		LD E, 04H	mengisi register E dengan data 04
	180A	60			LD H, B	menyalin isi register B ke register H
	180B	69			LD L, C	menyalin isi register C ke register L
	180C	31	9F	**1F**	LD SP,1F9FH	mengisi stack pointer 1F9F
	180F	FF			Halt	menutup program

- **Hasil**

Register A = 00H, B = 01H, C = 02H, D = 03H, E = 04H

H = 01H, L = 02H, SP = 1F9FH

b. Masalah 2:

Menyalin isi register B dan C ke alamat penyimpan data (RAM) alamat 1A00 H dan 1A01 H.

Register B = EEH

Register C = DDH

Program dimulai pada alamat 1900

Penyelesaian.

- **Struktogram**

Mengisi register B = EEH
Mengisi register C = DDH
Mengisi register HL = 1A00H
Menyalin isi register B pada alamat yang ditunjuk register HL
Mengisi register HL = 1A01H
Menyalin isi register C pada alamat yang ditunjuk register HL
Mengakhiri program

➢ **Program**

Label	Alamat (HEX)	Kode program Op-Code			Mnemonik	Keterangan
	1900	06	EE		LD B, EEH	mengisi register B dengan data EE
	1902	0E	DD		LD C,DDH	mengisi register C dengan data DD
	1904	21	00	1A	LD HL,1A00H	mengisi register HL dengan data 1A00
	1907	70			LD (HL), B	menyalin isi register B ke alamat yang ditunjuk register HL
	1908	21	01	1A	LD HL, 1A01H	mengisi register HL dengan data 1A01
	190B	71			LD (HL), C	menyalin isi register C ke alamat yang ditunjuk register HL
	190C	FF			Halt	menutup program

➢ Hasil

Register B = **EEH**

Register C = **DDH**

Alamat 1A00H = **EEH**

Alamat 1A01H = **DDH**

c. Masalah 3 :

Mengisi lokasi penyimpan data (RAM) yang alamatnya 1910H dengan data 56H dan 1911H dengan data 78H. Menyalin isi lokasi RAM alamat 1910H dan 1911H ke register D dan E.

Program dimulai pada alamat 1C00H.

Penyelesaian.

➢ **Struktogram**

Mengisi register HL dengan data 1910 H (alamat RAM)
Mengisi lokasi RAM yang alamatnya ditunjuk oleh HL dengan data 56H
Mengisi register HL dengan data 1911H (alamat RAM)
Mengisi lokasi RAM yang alamatnya ditunjuk oleh HL dengan data 78H
Mengisi register HL dengan data 1910H
Menyalin isi lokasi RAM yang ditunjuk oleh register HL ke register D
Mengisi register HL dengan data 1911H
Menyalin isi lokasi RAM yang ditunjuk oleh register HL ke register E
Mengakhiri program

➢ **Program**

Label	Alamat (HEX)	Kode program Op-Code			Mnemonik	Keterangan
	1C00	21	10	19	LD HL, 1910H	mengisi register HL dengan data 1910
	1C03	36	56		LD (HL), 56H	mengisi register (HL) dengan data 56
	1C05	21	11	19	LD HL,1911H	mengisi register HL dengan data 1911
	1C08	36	78		LD (HL), 78H	mengisi re gister (HL) dengan data 78
	1C0A	21	10	19	LD HL, 1910H	mengisi register HL dengan data 1910
	1C0D	56			LD D, (HL)	menyalin isi register (HL) ke register D
	1C0E	21	11	19	LD HL, 1911H	mengisi register HL dengan data 1911
	1C11	5E			LD E, (HL)	menyalin isi register (HL) ke register E
	1C12	FF			Halt	menutup program

➢ **Hasil**

Alamat RAM 1910H = **56 H**

Alamat RAM 1911H = **78 H**

Register D = **56 H**

Register E = **78 H**

d. Masalah 4 :

Mengisi alamat penyimpan data (RAM) 1B00H dengan data AAH.

Menyalin isi penyimpan data (RAM) alamat 1B00H ke alamat 1BFFH melalui akkumulator program dimulai pada alamat 1C00H.

Penyelesaian.

➢ **Struktogram**

Mengisi register HL dengan 1B00H (alamat RAM sumber data)
Mengisi alamat RAM yang alamatnya di tunjuk oleh HL dengan data AAH
Menyalin lokasi RAM yang alamatnya ditunjuk oleh HL ke akkumulator
Mengisi register HL dengan 1BFFH (alamat RAM tujuan data)
Menyalin isi akkumulator pada alamat yang ditunjuk oleh register HL

- **Program**

Label	Alamat (HEX)	Kode program Op-Code			Mnemonik	Keterangan
	1C00	21	00	1B	LD HL, 1B00H	mengisi register HL dengan data 1B00
	1C03	36	AA		LD (HL), AAH	mengisi register (HL) dengan data AA
	1C05	7E			LD A, (HL)	menyalin isi register (HL) ke akkumulator
	1C06	21	FF	1B	LD HL ,1BFFH	mengisi register HL dengan data 1BFF
	1C09	77			LD (HL), A	menyalin isi akkumulator) ke register (HL)
	1C0A	FF			Halt	menutup program

- **Hasil**

Isi alamat RAM 1B00H = **AA H**

Isi alamat RAM 1BFFH = **AA H**.

Isi akkumulator = **AA H**

e. Masalah 5 :

Mengisi lokasi RAM alamat 1E05H, 1E06H, 1E07H, masing-masing dengan data 01H, 02H dan 03H.Pengsian diatas mempergunakan pengalamatan terindeks dengan alamat offset 1E00H.

Program dimulai pada alamat 1D00H.

Penyelesaian.

➢ **Struktogram**

isi register IX dengan data 1E00H (alamat RAM sebagai offset)
isi lokasi RAM yang beralamat offset + 5 = (IX + 5) = 1E05H dengan data 01H
isi lokasi RAM yang beralamat offset + 6 = (IX + 6) = 1E06H dengan data 02H
isi lokasi RAM yang beralamat offset + 7 = (IX + 7) = 1E07H dengan data 03H
mengakhiri program

➢ **Program**

Label	Alamat (HEX)	Kode program Op-Code			Mnemonik	Keterangan
	1D00	DD	21	001E	LD 1X, 1E00H	mengisi register IX dengan data alamat offset
	1D04	DD	36	0501	LD (IX+5),01H	isi alamat offset tambah 5 dengan data 01
	1D08	DD	36	0602	LD (IX+6),02H	isi alamat offset tambah 6 dengan data 02
	1D0C	DD	36	0703	LD (IX+7),03H	isi alamat offset tambah 7 dengan data 03
	1D10	FF			Halt	menutup program

➢ **Hasil**

Lokasi RAM alamat 1E05H = **01H**

Lokasi RAM alamat 1E06H = **02H**

Lokasi RAM alamat 1E07H = **03H**

2. Program Aritmatik

a. Masalah 1 :

Mengisi akkumulator dengan data FEH tambahkan isi AKKU dengan data langsung 02H (tanpa melewati register), kemudian hasilnya bersama carry di kurangkan dengan data langsung 0AH.

Program dimulai pada alamat 1800 H.

Penyelesaian :

➢ **Struktogram**

Mengisi akku dengan data FEH
Tambahkan isi akkur dengan data langsung 02H
Kurangkan isi akku bersama carry dengan data langsung 0AH
Mengakhiri program

➢ **Program**

Label	Alamat (Hex)	Kode Operasi Op - Code			Mnemonik	Keterangan
	1800	3E	FE		LD A, FEH	Mengisi reg.A dengan data FEH
	1802	C6	02		ADD A, 02H	Menambah langsung reg.A dengan data 02H
	1804	DE	0A		SBC A, 0AH	Mengurangi carry, isi reg. A dengan data 0AH
	1805	FF			HALT	Mengakhiri program

➢ **Hasil**

Akkumulator : F5H

b. Masalah 2 :

Mengisi akkumulator dengan data 15H, register D dengan data E AH dan register B dengan data 55H.

Tambahkan isi akku dengan isi register D.

Hasil penjumlahan diatas bersama carrynya di kurangi dengan isi register B.

Program dimulai pada alamat 1900 H.

Penyelesaian

➢ **Struktogram**

isi akhir dengan data 15H
isi register D dengan data EAH
isi register B dengan data 55H
tambahkan isi akku dengan isi register D
kurangi isi akku bersama carry dengan isi register B
mengakhiri program

➢ **Program**

Label	Alamat (Hex)	Kode Operasi Op - Code			Mnemonik	Keterangan
	1900	3E	15		LD A, 15H	Mengisi reg A dengan data 15H
	1902	16	EA		LD D, EAH	Mengisi reg D dengan data EA H
	1904	06	55		LD B, 55H	Mengisi reg B dengan data 55H
	1906	82			ADD A, D	Menambah isi akku dengan isi reg D
	1907	98			SBC A, B	Mengurangi isi akku dengan reg.B dan carry
	1908	FF			HALT	Mengakhiri program

- **Hasil**

  Akkumulator = AA H

  Register D = EA H

  Register B = 55 H

c. Masalah 3 :

Mengisi lokasi RAM alamat 1F00H = 01H, 1F01H = 0FH dan 1F02H = 7BH

Kurangkan isi lokasi RAM alamat 1F00H dengan isi lokasi RAM alamat 1F01 H.

Hasil pengurangan diatas bersama carrynya di tambahkan dengan isi lokasi RAM alamat 1F02H.

Program dimulai pada alamat 1A00H

Penyelesaian :

- **Struktogram**

isi register HL dengan data 1F02H (alamat RAM)
isi lokasi RAM yang alamatnya ditunjuk oleh HL dengan data langsung 7B H
isi register HL dengan data 1F01 H (alamat RAM)
isi lokasi RAM yang alamatnya ditunjuk oleh HL dengan data langsung 0F H
isi register HL dengan data 1F00H (alamat RAM)
isi lokasi RAM yang alamatnya ditunjuk oleh HL dengan data langsung 01 H
menyalin isi lokasi RAM yang alamatnya ditunjuk oleh HL ke akku
isi register HL dengan data 1F01 H (alamat RAM)
kurangi akhir dengan isi lokasi RAM yang alamatnya ditunjuk oleh HL
isi register HL dengan data 1F02 H (alamat RAM)
tambahkan akhir bersama carry dengan isi lokasi RAM yang alamatnya ditunjuk oleh H6
mengakhiri program

- **Program**

Label	Alamat (Hex)	Kode Operasi Op - Code			Mnemonik	Keterangan
	1A00	21	02	1F	LD HL,1F02H	Mengisi reg HL dengan data 1F02H
	1A03	36	7B		LD (HL),7BH	Mengisi lokasi RAM yang alamatnya ditunjuk HL dengan data langsung 7BH
	1A05	21	01	1F	LD HL,1F01H	Mengisi reg HL dengan data 1F01
	1A08	36	0F		LD (HL),0FH	Mengisi lokasi RAM yang alamatnya ditunjuk oleh HL dengan data 0F
	1A0A	21	00	1F	LD HL,1F00H	Mengisi reg HL dengan data 1F00
	1A0D	36	01		LD (HL),01H	Mengisi lokasi RAM yang alamatnya ditunjuk oleh HL dengan data 01
	1A0F	7E			LD A,(HL)	Menyalin isi HL ke akkumulator
	1A10	21	01	1F	LD HL,1F01H	Mengisi reg HL dengan data 1F01
	1A13	96			SUB A,(HL)	Mengurangi A dengan data pada lokasi HL
	1A14	21	02	1F	LD HL,1F02H	Mengisi HL dengan data 1F02
	1A17	8E			ADC A, (HL)	Menambah bersama carry reg A dengan data pada HL
	1A18	FF			HALT	Mengakhiri program

- **Hasil**

Lokasi RAM alamat 1F00H = 01 H

Lokasi RAM alamat 1F01H = OF H

Lokasi RAM alamat 1F02H = 7B H

Akkumulator = 6E H

d. Masalah 4 :

Mengisi lokasi RAM alamat 1E07, 1E08H dan 1E09H masing-masing dengan data 01H, 02H dan 03H.

(pengisian mempergunakan pengalamatan terindeks).

Tambahkan isi lokasi RAM yang beralamat 1E07H dengan isi alamat 1E09H. (menggunakan pengalamatan terindeks). Alamat offset 1E00H.

Program dimulai pada alamat 1F00H

Penyelesaian

- **Struktogram**

isi register 1X dengan alamat offset 1E00 H
isi lokasi RAM alamat offset +7 = 1E07 H dengan data 01 H
isi lokasi RAM alamat offset + 8 = 1E08 H dengan data 02 H
isi lokasi RAM alamat offset + 9 = 1E08 H dengan data 03 H
salin isi lokasi RAM alamat offset + 7 = 1E07 H pada akku
tambahkan isi akku dengan lokasi RAM alamat offset + 9
mengakhiri program

- **Program**

Label	Alamat (Hex)	Kode Operasi Op - Code			Mnemonik	Keterangan
	1F00	DD	21	001E	LD IX,1E00H	Mengisi alamat offset dengan data 1E00H
	1F04	DD	36	0701	LD(IX+7),01H	Mengisi alamat offset+7 dengan data 01 H
	1F08	DD	36	0802	LD(IX+8),02H	Mengisi alamat offset+8 dengan data 02 H
	1F0C	DD	36	0903	LD(IX+9),03H	Mengisi alamat offset+9 dengan data 03 H
	1F10	DD	7E	07	LD A,(IX+7)	Menyalin isi alamat offset+7 ke akku
	1F13	DD	86	09	ADD A,(IX+9)	Menambah akku dengan data pada alamat offset +9
	1F16	FF			HALT	Mengakhiri program

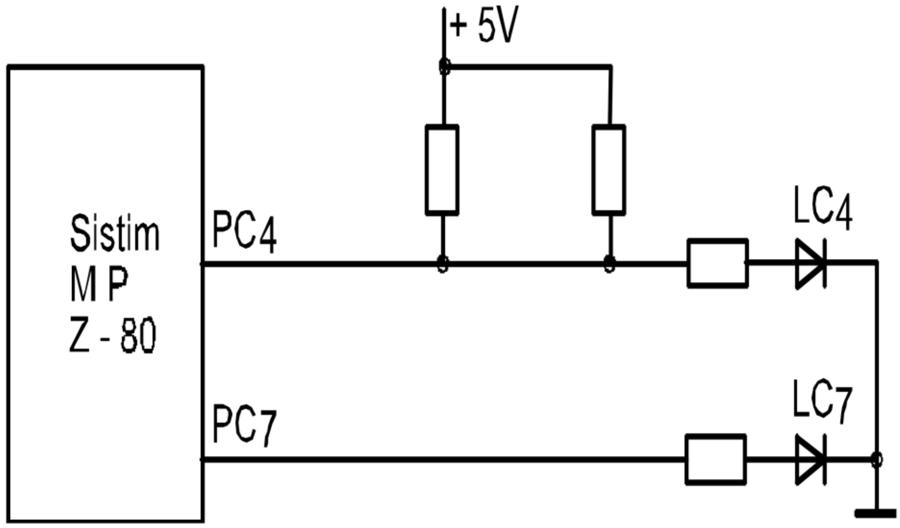
- **Hasil**

Lokasi RAM alamat 1E07H	=	01 H
Lokasi RAM alamat 1E08H	=	02 H
Lokasi RAM alamat 1E09H	=	03 H
Akkumulator	=	04 H

3. Program Masukan Keluaran

a. Masalah 1 :

Mengeluarkan data FFH dan 00H dengan tunda waktu FFH x FFH program berlangsung terus, sampai pada penekanan tombol reset. Program utama pada 1800H Program bagian pada 1F00H



Alamat register kontrol 43 H

Alamat Port C 42 H

Kata kendali : 80 H (Port A, B dan C = keluaran)

Penyelesaian :

➢ **Struktogram**

PROGRAM UTAMA

- Mengisi akku dengan data kata kendala (80 H)
- Keluarkan isi akku ke register control
- loop (pengulangan tak berakhir)
    - isi akku dengan data FFH
    - keluarkan isi akku ke register port C tinggi
    - tunda waktu
    - isi akku dengan data 00 H
    - keluarkan isi akku ke register port C tinggi
    - tunda waktu

PROGRAM BAGIAN TUNDA WAKTU

- isi register H dengan data FFH
- loop 1
    - isi register L dengan data FFH
    - loop 2
        - kurangkan isi L dengan 1
        - loncat ke loop 2, bila LF 0
    - Kurangkan isi H dengan 1
    - loncat ke loop 1, bila HF 0
- Mengakhiri program bagian

- **Program**

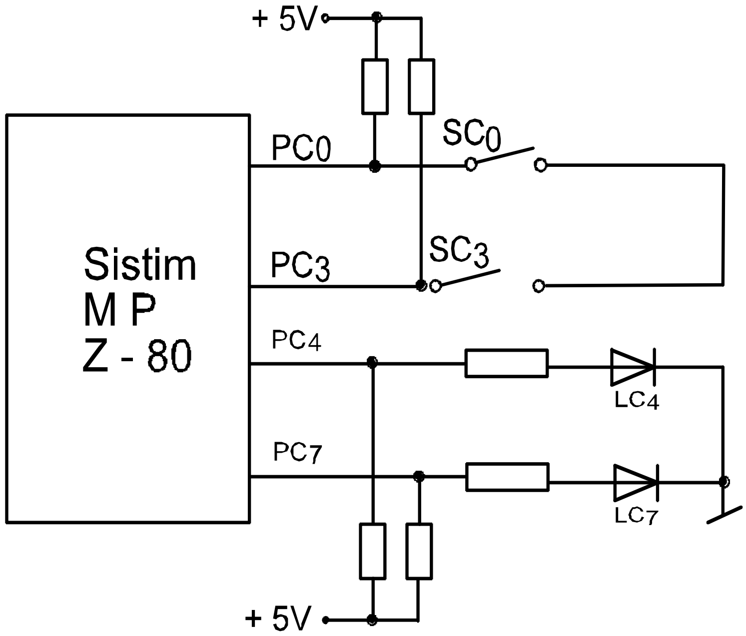
Label	Alamat (Hex)	Op - Code			Mnemonik	Keterangan
	1800	3E	80		LDA, 80 H	- mengisi akku dengan data 80H
	1802	D3	43		OUT 43, A	- mengeluarkan isi akku ke reg control alamat 43H
loop	1804	3E	FF		LDA , FF H	- mengisi akku dengan data FFH
	1806	D3	42		OUT 42, A	- mengeluarkan isi akku ke port C tinggi alamat 42H
	1808	CD	00	1F	CALL UNC TW	- memanggil tanpa syarat tunda waktu
	180B	3E	00		LDA , 00 H	- mengisi akku dengan data 00H
	180D	D3	42		OUT 42, A	- mengeluarkan isi akku ke port C tinggi alamat 42H
	180F	CD	00	1F	CALL UNC TW	- memanggil tanpa syarat tunda waktu
	1812	C3	04	18	LDA, 00H	- loncat tanpa syarat ke alamat 1804H
TW	IF00	26	FF		LDH, FFH	- mengisi reg H dengan data FF H
	IF02	2E	FF		LDL, FFH	- mengisi reg L dengan data FF H
	IF04	2D			DEC, L	- isi reg L dikurangi 1
	IF05	C2	04	IF	JPNZ, 1F04	- loncat ke alamat 1F04 bila CF0 (Z = 0)
	IF06	25			DEC, H	- isi reg H dikurangi 1
	IF09	C2	02	IF	JPNZ, 1F02	- loncat ke alamat 1F02 bila CF0 (Z = 0)
	IF0C	C9			Ret UNC	- mengulang tanpa syarat

- **Hasilnya**

Lampu berkedip-kedip terus dan berhenti saat tombol reset ditekan

b. Masalah 2 :

Membaca data dari port C sendok dan mengeluarkan data tersebut pada register C tinggi.



Alamat kata kendali : 43 H

Alamat Port C 42 H

Program dimulai pada alamat 1800 H

Program berlangsung terus sampai pada penekanan reset.

Penyelesaian :

➢ **Struktogram**

Isi akku dengan data kata kendala (91 H)	
keluarkan isi akku ke register kontrol	
loop (pengulangan tak berakhir)	
	baca isi port C rendah
	geser isi port C kekiri sebanyak 4 kali SLA A
	keluarkan isi akku ke port C tinggi

➢ **Program**

Label	Alamat	Op - Code			Mnemonik	Keterangan
	1800	3E	91		LDA, 91 H	- mengisi akku dengan data kata kendala 91
	1802	D3	43		OUT 43, A H	- keluarkan isi akku ke reg control
loop	1804	DB	42		IN A, 42 H	- baca port C rendah
	1806	CB	27		SLA , A	- geser isi port C ke kiri
	180A	CB	27		SLA , A	- geser isi port C ke kiri
	180C	CB	27		SLA , A	- geser isi port C ke kiri
	180E	D3	42		OUT 42 A H	- keluarkan isi akku ke Port C tinggi
	1810	C3	04	18	JPUNC loop	- loncat tanpa syarat ke alamat 1804

➢ **Hasilnya**

lampu bergeser terus dan saat tombol reset ditekan lampu nyala semua (berhenti)

c. Masalah 3 :

Membuat LED berjalan dari kiri ke kanan (dari bit PC0 ke PC3), dengan tanda waktu FFH x FFH.

Alamat register kontrol 43H

Alamat port C 42H

Program utama mulai pada alamat 1800 H

Program bagian tunda waktu pada alamat 1F00H

PC7 PC0

Penyelesaian :

- **Struktogram**

- isi akkumulator dengan 80 H
- keluarkan isi akkumulator ke register kontrol
- loop (pengulangan tidak berakhir)
  - isi akkumulator dengan data 10 H
  - isi register C dengan data 04 H
  - loop1
    - keluarkan isi akku ke port C tinggi
    - tunda waktu
    - geser isi akku ke kiri
    - kurangi isi C dengan 1
- loncat ke loop selama C ≠ 0

- **Program**

Label	Alamat	Op - Code			Mnemonik	Keterangan
	1800	3E	80		LDA, 80 H	- mengisi register A dengan data 80H
	1802	D3	43		OUT 43, A	- mengeluarkan isi akku ke reg control
loop	1804	3E	10		LDA , 10 H	- Isi akku dengan data 10
	1806	0E	04		LDC, 04 H	- mengisi register C dengan data 04
	1808	D3	42		OUT 40, A	- mengeluarkan isi akku ke port C
	180A	CD	00	1F	CAAL UNC TW	- loncat ke loop
	180D	CB	27		SLA, A	- isi akku digeser ke kiri
	18	OD			DEC, C	- mengurangi isi register C dengan 1
	181	C2	08	18	JPNZ, loop 1	- loncat bersyarat Z = 0 ke loop 1
	1813	C3	04	18	JPUNC, loop	- loncat tanpa syarat ke loop

- **Hasilnya**

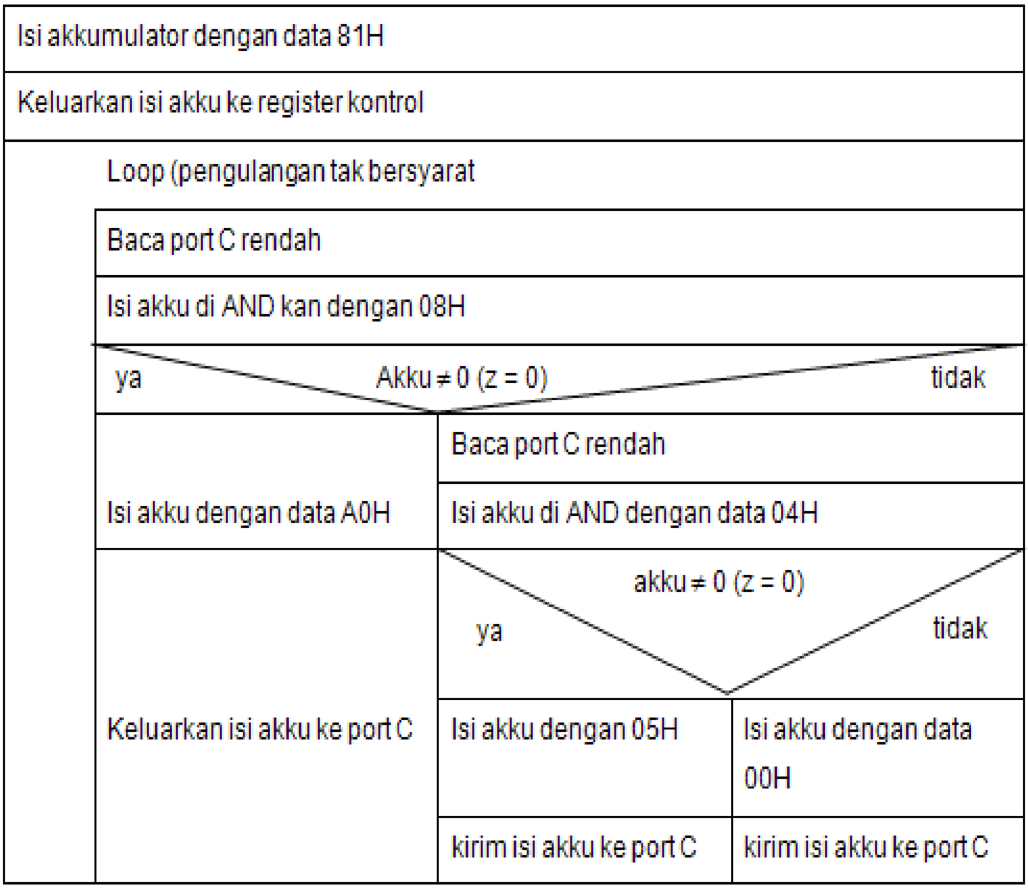
Lampu bergeser terus dan saat tombol reset ditekan lampu nyala semua (berhenti).

d. Masalah 4 :

Mengontrol LED (PC3 - PC0) oleh saklar (PC7 - PC4). Bila saklar PC7 = ON dan saklar yang lain OFF, LED PC3 dan PC1 = ON. Bila saklar PC6 = ON dan saklar yang lain OFF, LED PC2 dan PC0 = ON. Bila kondisi saklar diluar kedua kondisi diatas, maka semua LED program. Program dimulai pada alamat 1800, dan berlangsung terus sampai penekanan tombol direset.

Penyelesaian :

➢ **Struktogram**



➢ **Program**

Label	Alamat	Op - Code			Mnemonik	Keterangan
	1800	3E	81		LDA, 81 H	- mengisi akku dengan data 81H
	1802	D3	43		OUT 43, A	- mengeluarkan akku ke reg control
	1804	DB	42		In A, 42	- membaca port C
	1806	EG	08		And A, 08 H	- meng-and-kan akku dengan 08
	1808	C2	19		JPN 2, 1819	- meloncat dengan syarat Z=0 ke alamat 1819
	180B	DB	42		In A, 42	- membaca port C
	180D	EG	04		And A, 04	- meng-and-kan akku dengan 04
	180F	C2	20		JPNZ, 1820	- meloncat dengan syarat Z=0 ke alamat 1820
	1812	DB	00		LDA, 00	- mengisi akku dengan data 00
	1814	E6	42		Out, 42	- mengeluarkan isi akku dengan data A0
	1816	C2	04		JPNZ, 1804	- mengeluarkan akku ke port C
	1819	3E	00		LDA, 50	- meloncat dengan syarat Z=1 ke alamat 1804
	181B	D3			Out, 42	- mengisi akku ke port C
	181D	C3			JPZ, 1804	- meloncat dengan syarat Z=1 ke alamat 1804
	1820	3E				- mengisi akku dengan data 50
	1822	D3				- mengeluarkan akku ke port C
	1824	C3				- meloncat dengan syarat Z=1 ke alamat 1804

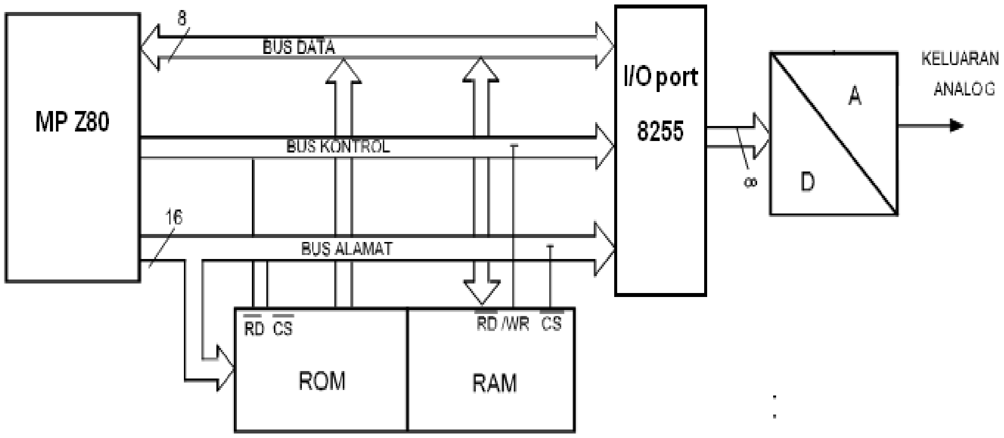
- **Hasilnya**

  Lampu bergeser terus dan saat tombol reset ditekan lampu nyala semua (berhenti).

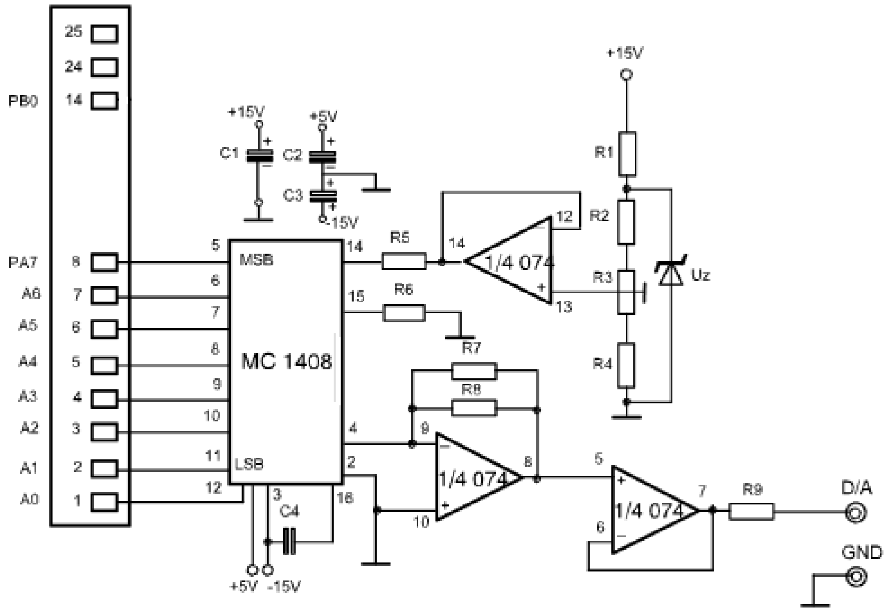
4. Pemrograman AD-DA Converter

a. *Digital To Analog Converter* (DAC)

Fungsi dari *digital to analog Converter* (DAC) adalah mengubah besaran digital yang diperoleh dari jalur data (*data bus*) untuk dijadikan besaran analog, selanjutnya besaran analog inilah yang digunakan untuk berbagai keperluan misal membuat generator fungsi, pengendali analog dan keperluan yang lain.Berikut merupakan blok diagram sebuah DAC yang dipasangkan pada sistem mikroprosesor Z80:



Berikut rangkaian elektronika dari pengalih digital ke analog, rangkaian didukung sebuah *chip* IC MC 1408 yang berfungsi sebagai DAC, dan rangkaian tegangan referensi yaitu melalui tegangan UZ diperoleh tegangan referensi yang stabil sehingga konversi sangat baik serta rangkaian penyangga beban untuk keluaran analog (op-amp 074).



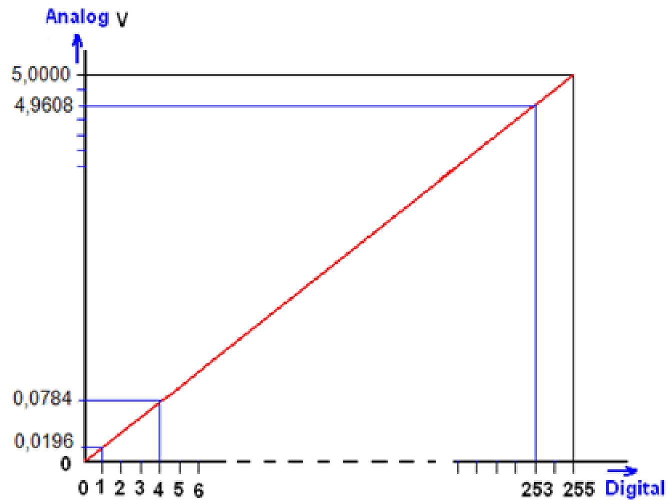
Prinsip kerja rangkaian yaitu rangkaian tersambung dengan port A dari IC 8255 sebagai keluaran untuk bus data masukan IC MC1408, dengan memberikan data 0 sampai 255 yang dikeluarkan melalui port A maka hasil konversi sudah dapat diukur paka keluarn D/A.

Untuk mikroprosesor Z80 memiliki bus data 8 bit, dengan adanya 8 bit tersebut memungkinkan menuliskan nilai digital 0000 0000 sampai 1111 1111. Jika tegangan referensi DAC adalah 5 volt maka kemungkinan besaran anlaog yang diperoleh dari hasil konversi adalah 0 volt sampai 5 volt DC, dan nilai digital yang bisa dikonversikan adalah 0 sampai 255. Sehingga dapai dicari nilai konversi setiap step atau setiap kenaikan 1 dari nilai digital adalah:

$$\Delta U = 1/255 \times 5 \text{ volt} = 0{,}0196 \text{ volt}$$

Data konversi digital ke analog adalah sebagai berikut:

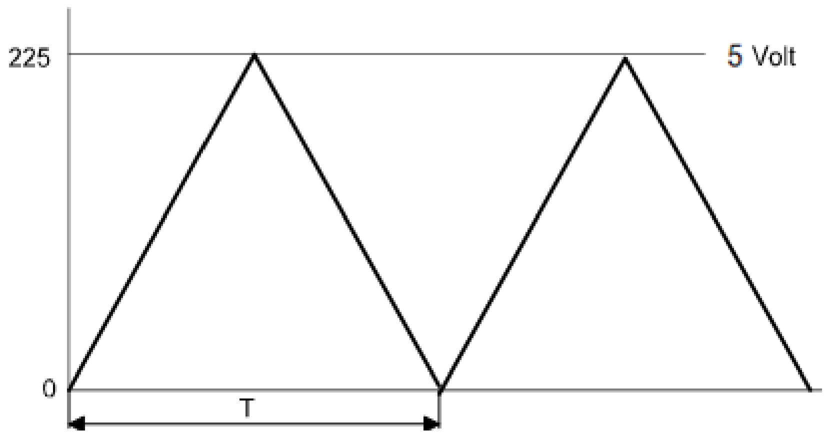
Nilai digital (biner)	Nilai analog (volt)
0000 0000	0.0000
0000 0001	0,0196
0000 0010	0,0392
0000 0011	0,0588
0000 0100	0,0784
1111 1101	4,9608
1111 1110	4,9804
1111 1111	5,0000



Masalah:

Buatlah program di mulai dari pembuatan struktogram yang dapat menghasilkan bentuk gelombang segi tiga seperti gambar dibawah ini :

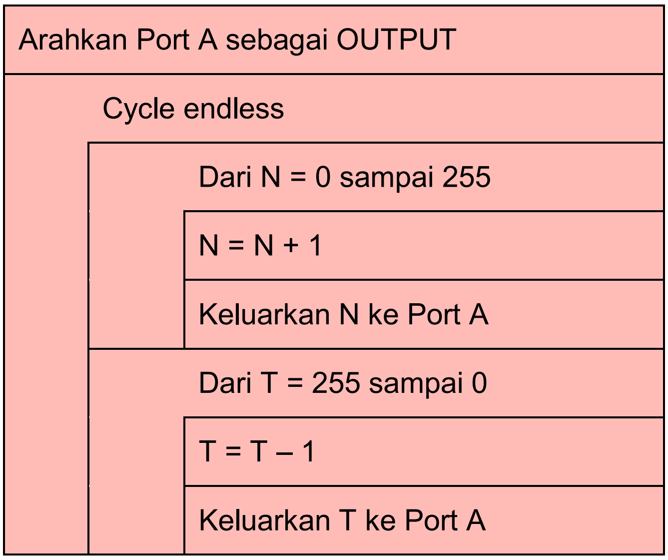
⇒ Bentuk gelombang di lihat dengan CRO



Penyelesaian:

Berdasarkan bentuk gelombang segi tiga didapatkan amplitudo maksimum 5 volt dan T = 2 x 255 (*counter*), dengan demikian maka terdapat perhitungan naik dari 0 ke 255 dengan penambahan 1 dan terdapat perhitungan turun dari 255 ke 0. Jika diharapkan gelombang bentuk segi tiga terus menerus ditampilkan pada keluaran analog, maka siklus atau loop dilakukan dengan tanpa batas.

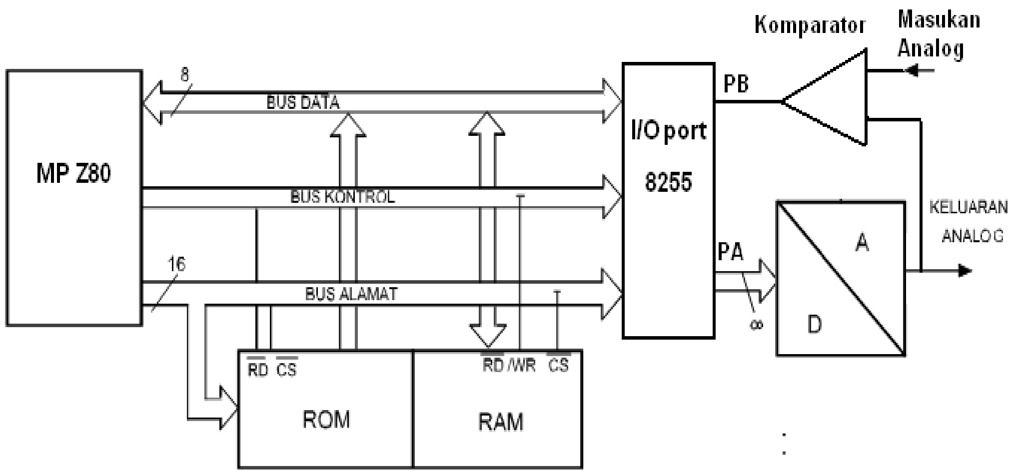
Struktogram



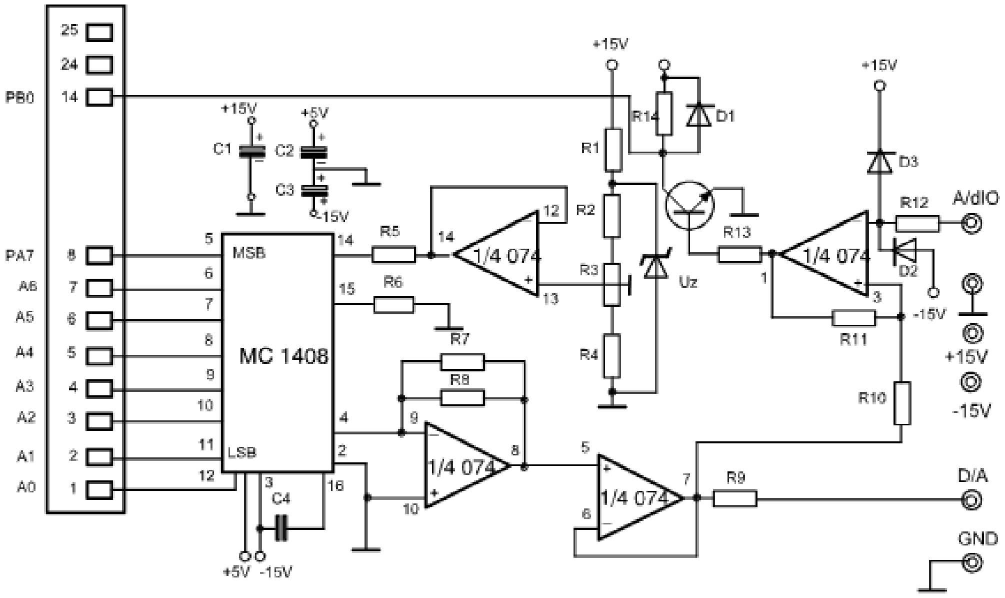
b. ADC (*Analog Digital Converter*)

Analog Digital Converter (ADC) secara bahasa diartikan pengalih analog ke digital, di dalamnya terdapat rangkaian elektronika yang fungsinya mengkonversikan sinyal analog menjadi sinyal digital. Sedangkan metode konversi meliputi servo ADC, successive approximation dan parallel converter, dalam rangkaian kali ini yang digunakan adalah pengubah sinyal analog menjadi sinyal digital proposional. Jenis ini adalah menggunakan metode pendekatan bertingkat (successive approximation convertion), disamping waktu yang relatif singkat dibanding metode yang lain proses konversi tidak mempengaruhi nilai masukan analog yang akan diubah.

Berikut blok diagram ADC yang terkoneksi dengan mikroprosesor Z80:



Rancangan ADC (*Analog Digital Con*verter) pada hakekatnya dibangun dari rangkaian DAC (Digital to Analog Converter) seperti ditunjukan pada gambar berikut:



Prinsip kerja rangkaian yaitu memadukan antara keluaran DAC dengan masukan besaran analok melalui rangkaian komparator, selanjutnya hasil komparator dihubungkan dengan port B dari port I/O 8255. Jika hasil komparasi 0 maka hasil komversi analog ke digital selesai dan selama hasil komparasi terbaca pada port B = 1 maka konversi masih harus dilakukan. Dengan memanfaatkan setiap kali

membaca logika 1 pada port B untuk menambah penghitung (counter) +1 maka penghitung akan terus menambah 1 sampai logika pada port B terbaca =0.

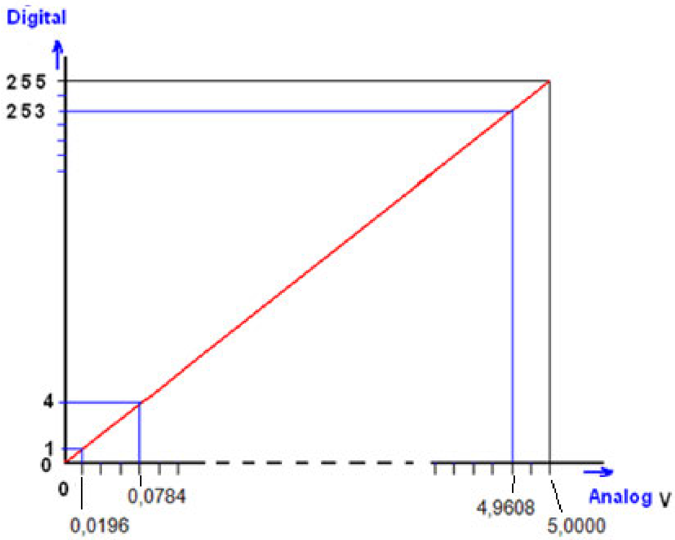
Di sisi lain rangkaian tersambung dengan port A dari IC 8255 sebagai keluaran untuk bus data masukan IC MC1408, dengan memberikan data 0 sampai 255 secara bertahap dan dikeluarkan melalui port A maka hasil konversi keluaran D/A akan naik. Dengan kenaikan nilai analog ini selalu dibandingkan dengan masukan anlog yang dikonversikan, jika tegangan masukan analog lebih rendah maka konversi berakhir karena pada port B akan terbaca = 0 .

Jika tegangan referensi DAC adalah 5 volt maka kemungkinan besaran anlaog yang diperoleh dari hasil konversi adalah 0 volt sampai 5 volt DC, dan nilai digital yang bisa dikonversikan adalah 0 sampai 255. Sehingga dapai dicari nilai konversi setiap step atau setiap kenaikan 1 dari nilai digital adalah:

$$\Delta U = 1/255 \times 5 \text{ volt} = 0{,}0196 \text{ volt}$$

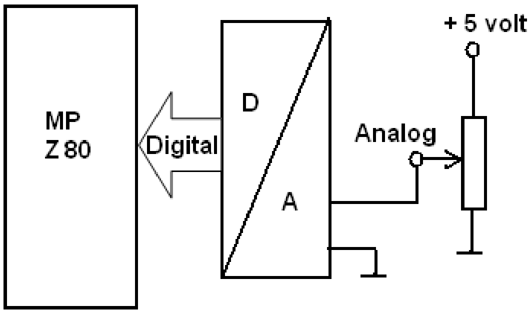
Data konversi analog ke digital adalah sebagai berikut:

Nilai analog (volt)	Nilai digital (biner)
0.0000	0000 0000
0,0196	0000 0001
0,0392	0000 0010
0,0588	0000 0011
0,0784	0000 0100
⋮	⋮
4,9608	1111 1101
4,9804	1111 1110
5,0000	1111 1111



Masalah:

Buatlah program di mulai dari pembuatan struktogram yang dapat mengkonversi besaran analog 0 sampai 5 volt menggunakian rangkaian dasar sebagai berikut:



Struktogram

Arahkan Port A sebagai output dan Port B sebagai input		
Inisialisasi penghitung N = 0		
Cycle endless		
	Port B = 1	
		N = N + 1
		Keluarkan N ke Port A
		Baca Port B
	Hasil konversi =N	

DAFTAR PUSTAKA

Microprocessor Architecture FROM SIMPLE PIPELINES TO CHIP MULTI PROCESSORS, Jean-Loup Baer, 2010

Understanding 8085/8086 Microprocessor and Peripheral IC′s Through Questions and Answers (Second Editions), S.K. Sen, 2010, Visit us at www.newagepublishers.com

Analog Interfacing to Embedded Microprocessor Systems, Stuart R. Ball, 2004

Microprocessor Design A Practical Guide from Design Planning to Manufacturing, Grant McFarland, 2006

Microprocessor Design Principles and Practices With VHDL, Enoch O. Hwang, 2004